SEHH2239 Data Structures
Lecture 10

Learning Objectives:

- To describe the Min / Max Trees
- To understand Heap structure and implement it in array
- To put and remove items in Heap
- To convert data to heap by heapifying and its performance
- To implement the priority queue
- To sort data by heapsort

Heap and Heap Sort

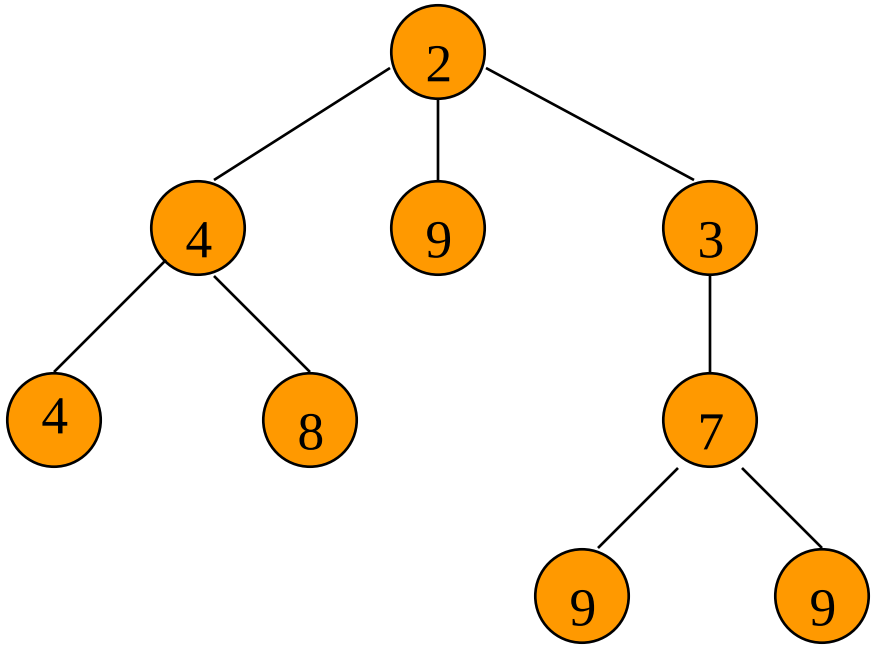
Two kinds of heap:

- Min Heap.
- Max Heap.

Min Tree Definition

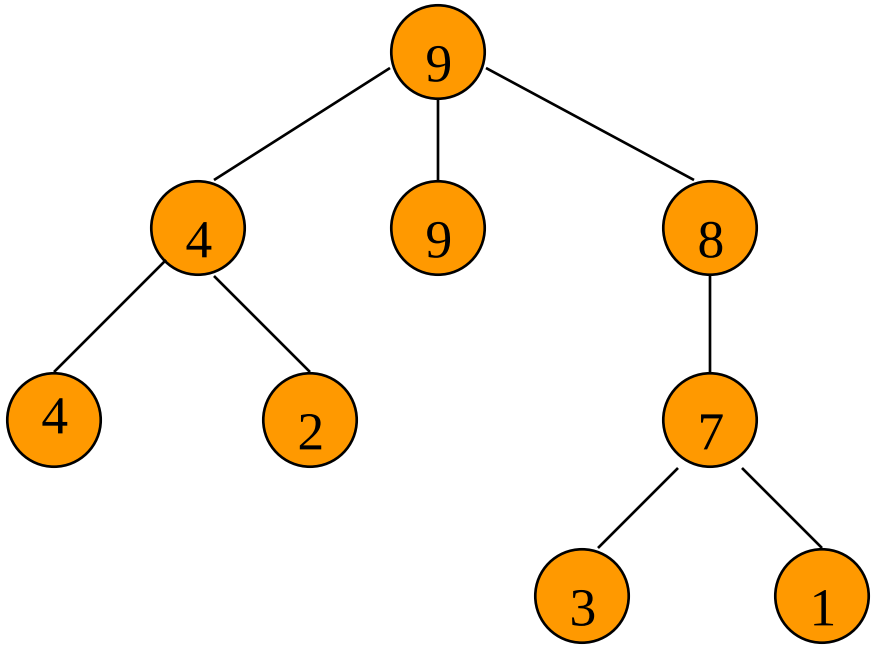
- Each tree node has a value.
- Value in any node is the **minimum value** in the subtree for which that node is the **root**.
- Equivalently, no descendent has a smaller value.

Min Tree Example



Root has minimum element.

Max Tree Example

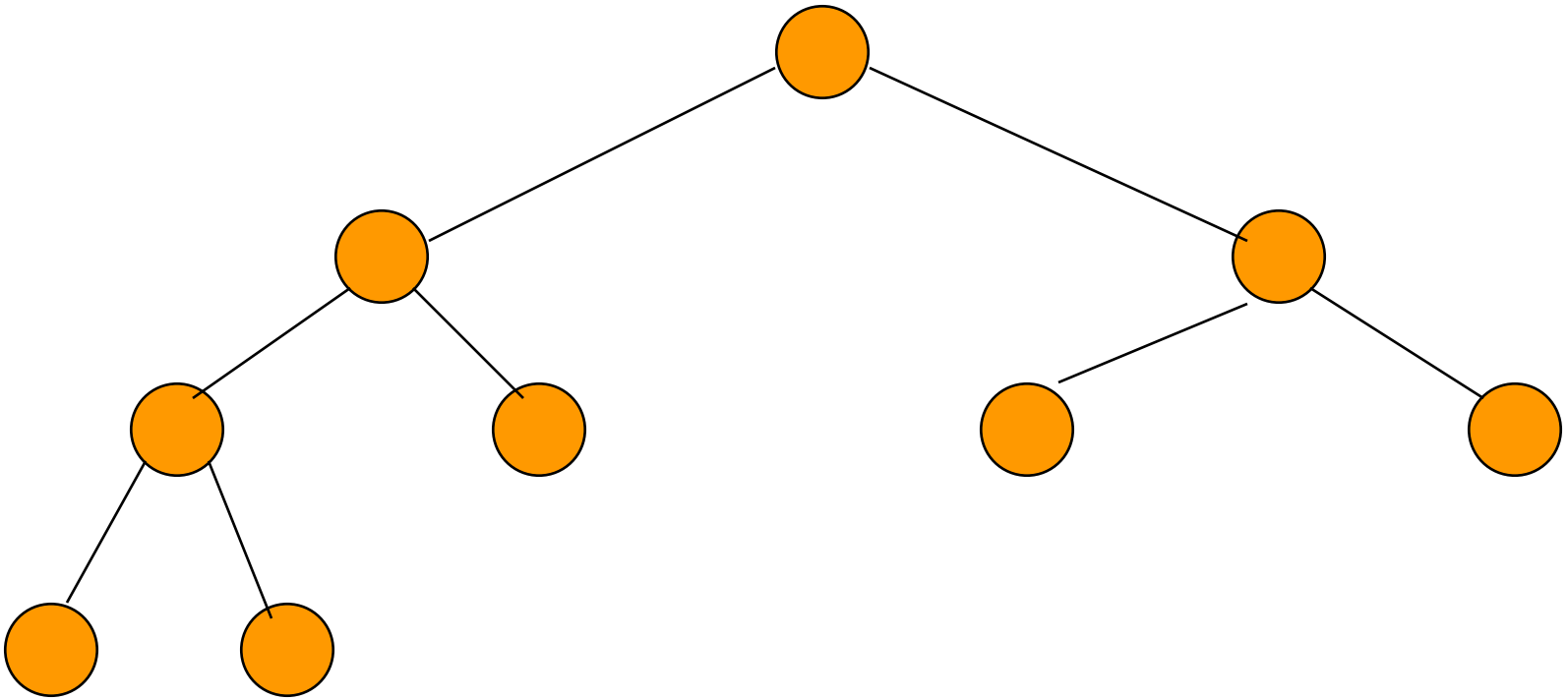


Root has maximum element.

Min Heap Definition

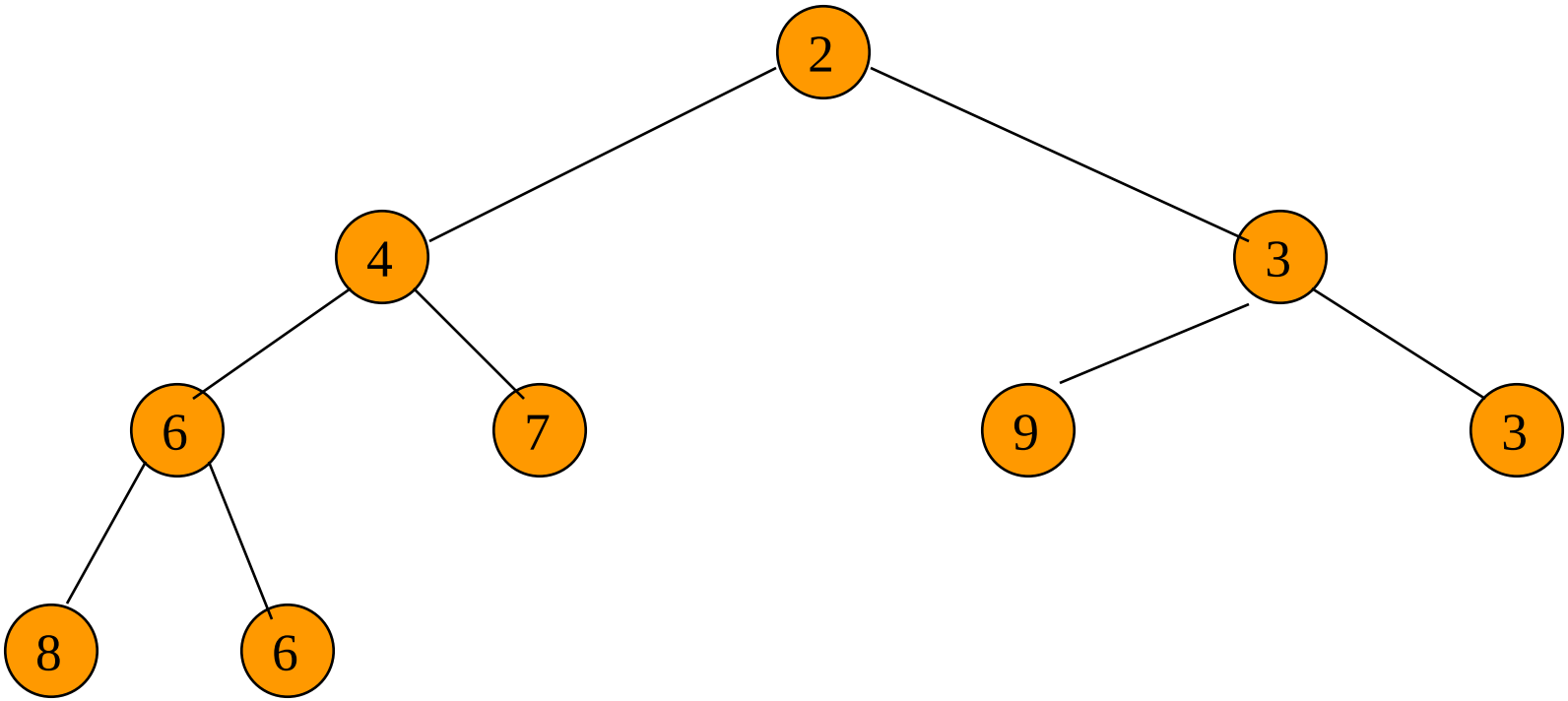
- **complete** binary tree
- **min** tree

Min Heap With 9 Nodes



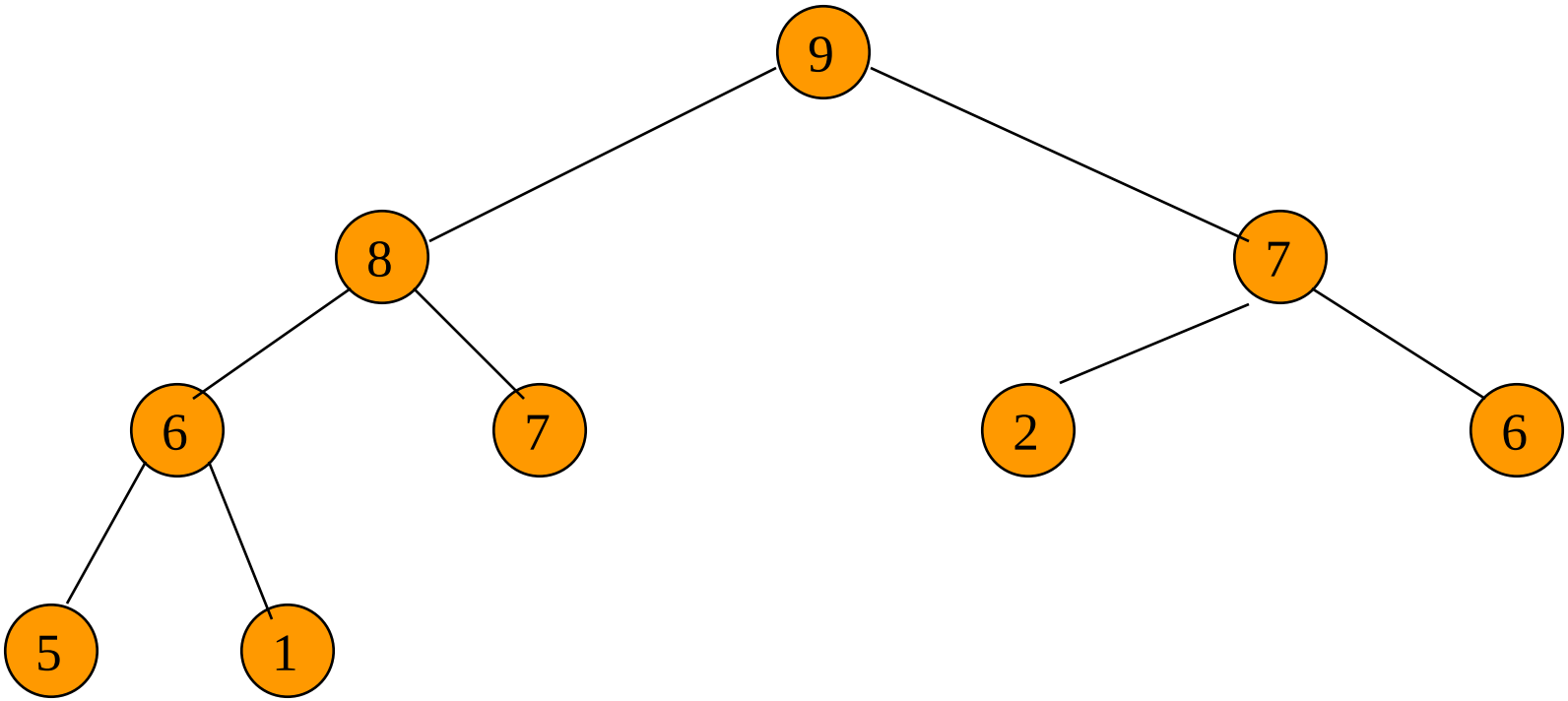
Complete binary tree with 9 nodes.

Min Heap With 9 Nodes



Complete binary tree with 9 nodes that is also a min tree.

Max Heap With 9 Nodes



Complete binary tree with 9 nodes that is also a max tree.

Heap Height

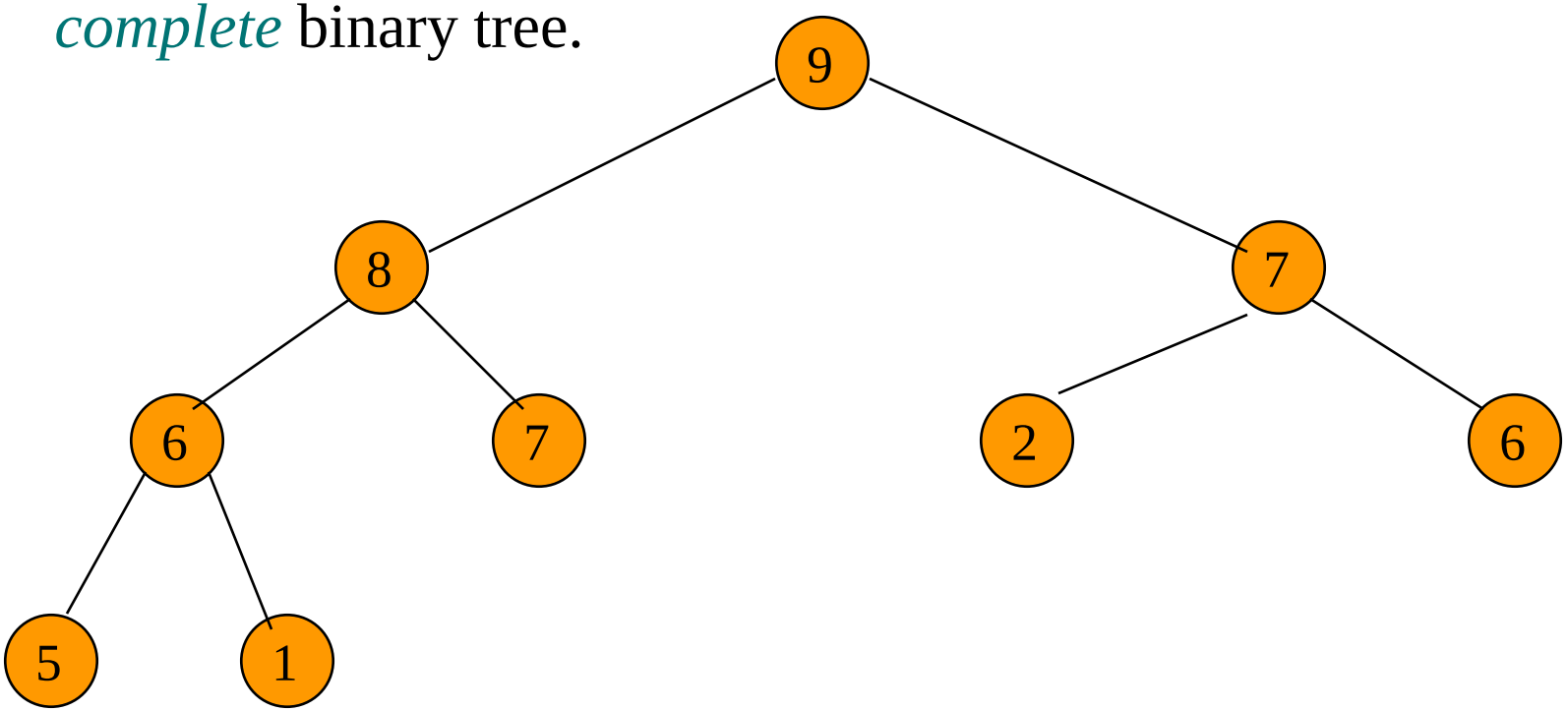
- By default, a **heap** is a **max heap**; a **min heap** must be explicitly specified.
- Since a heap is a complete binary tree, the height of an **n** node heap is

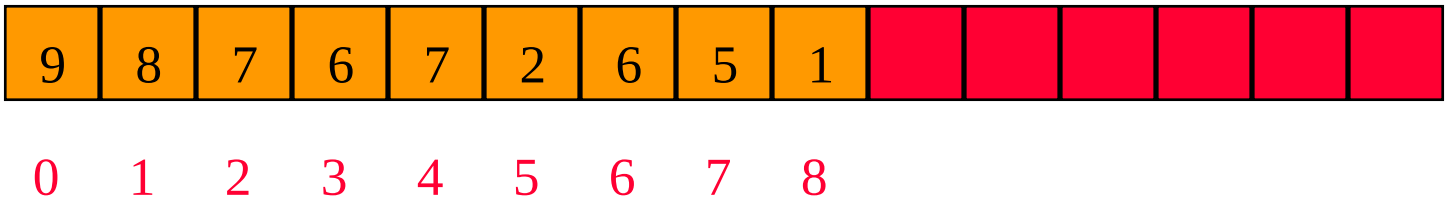
$$\lfloor \log_2 n \rfloor + 1$$

Heap implemented in An Array

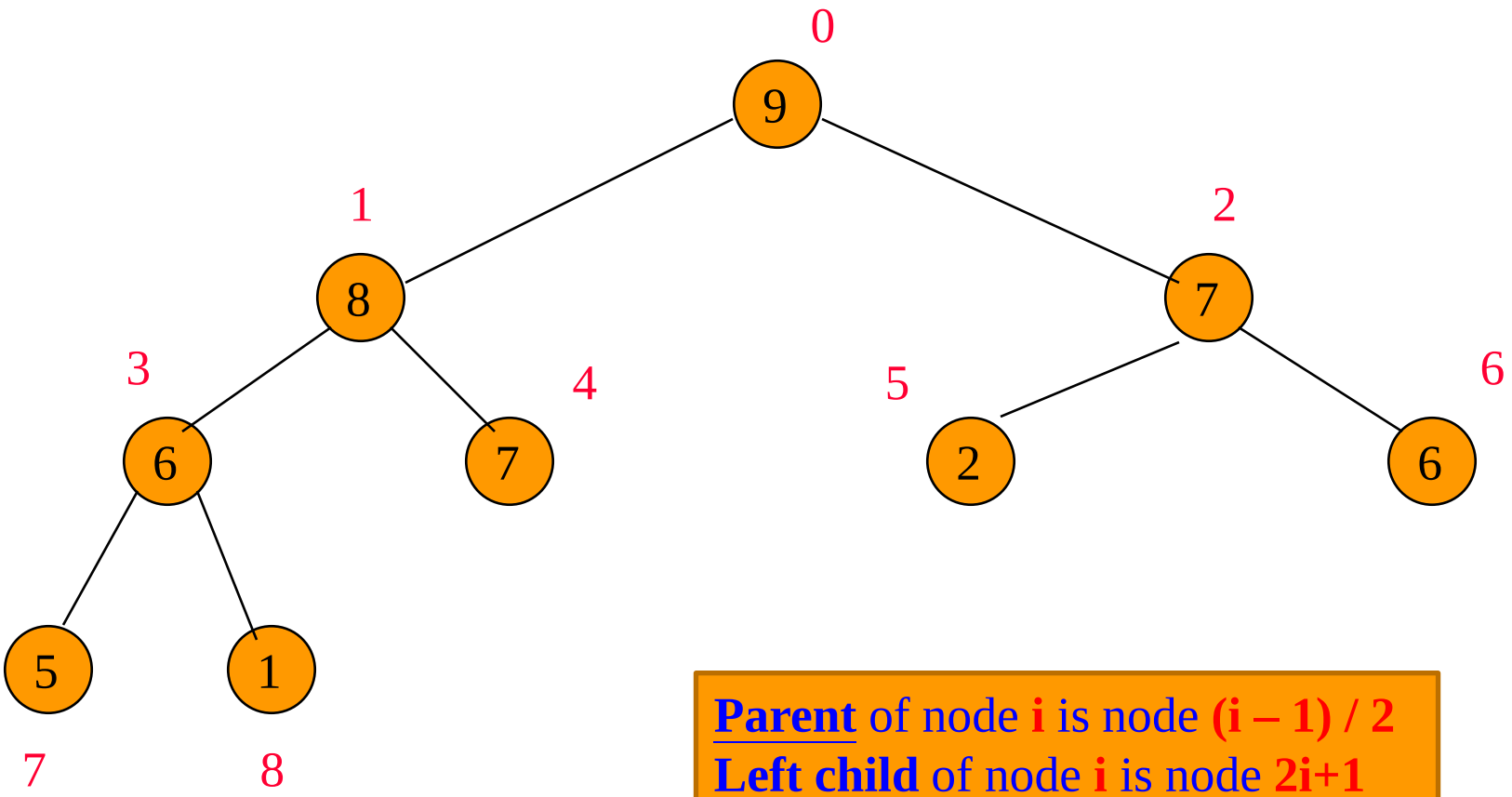
A Heap Represented As An Array

- A Heap Is Efficiently Represented in an array as a *complete* binary tree.





Moving Up And Down A Heap

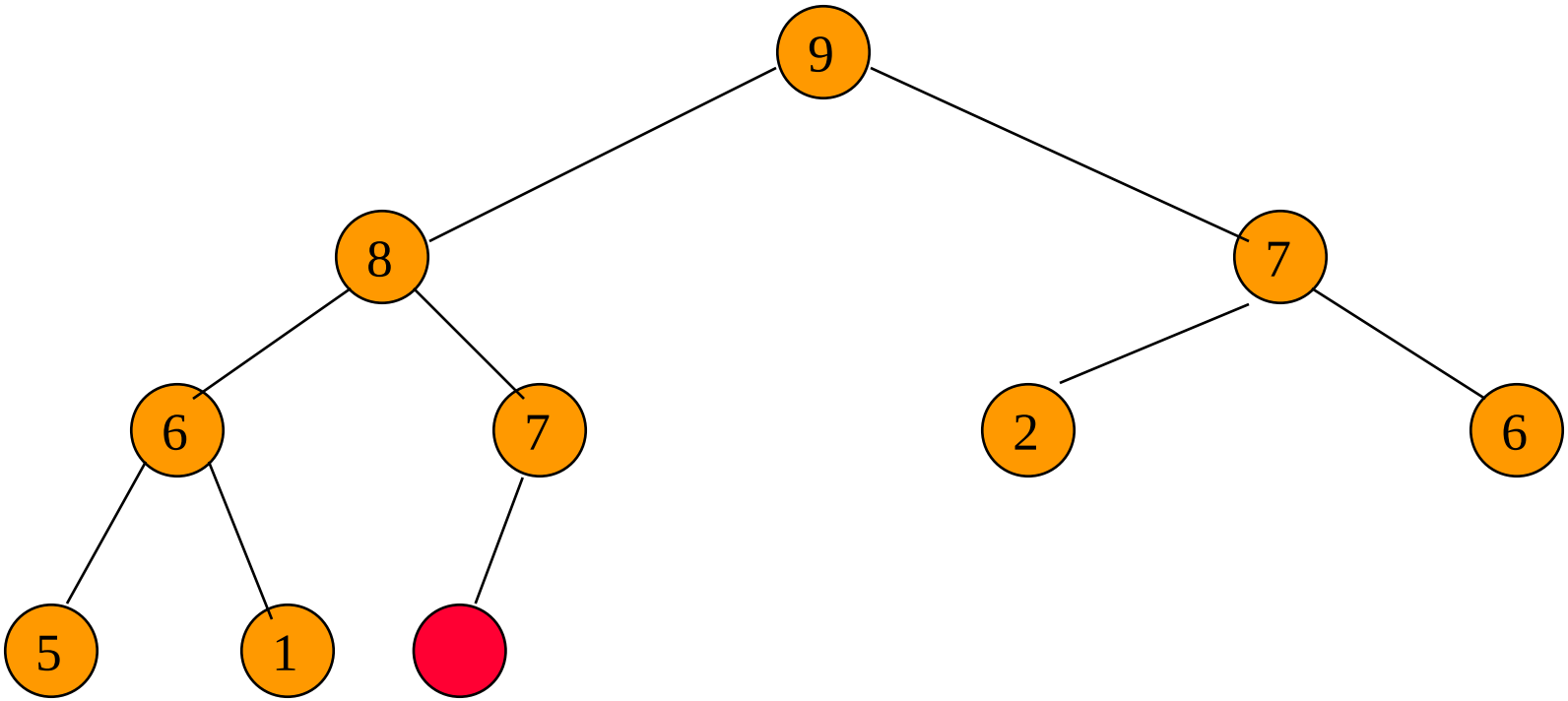


Parent of node **i** is node **(i – 1) / 2**
Left child of node **i** is node **2i+1**
Right child of node **i** is node **2i+2**

Implementation of Heap - PUT

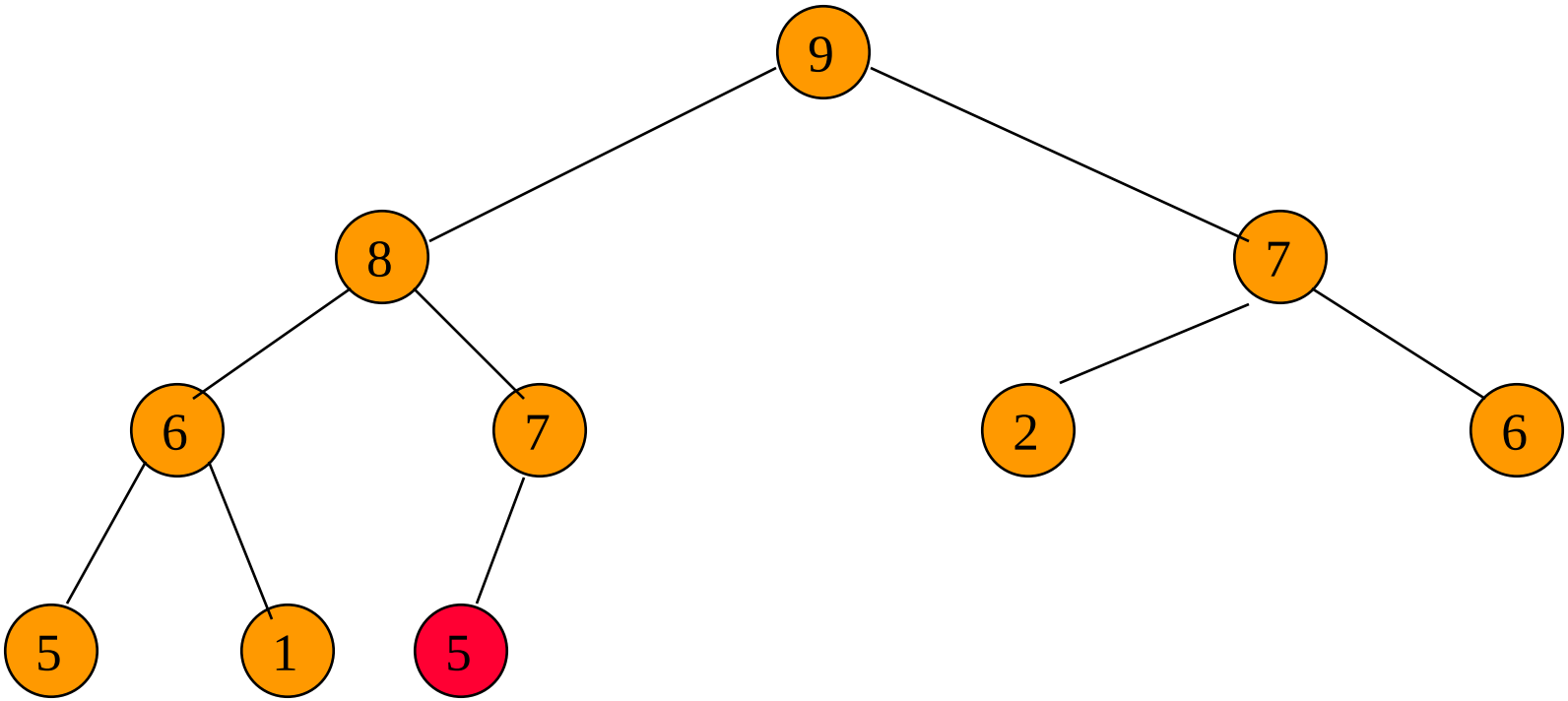
15

Putting An Element Into A Max Heap



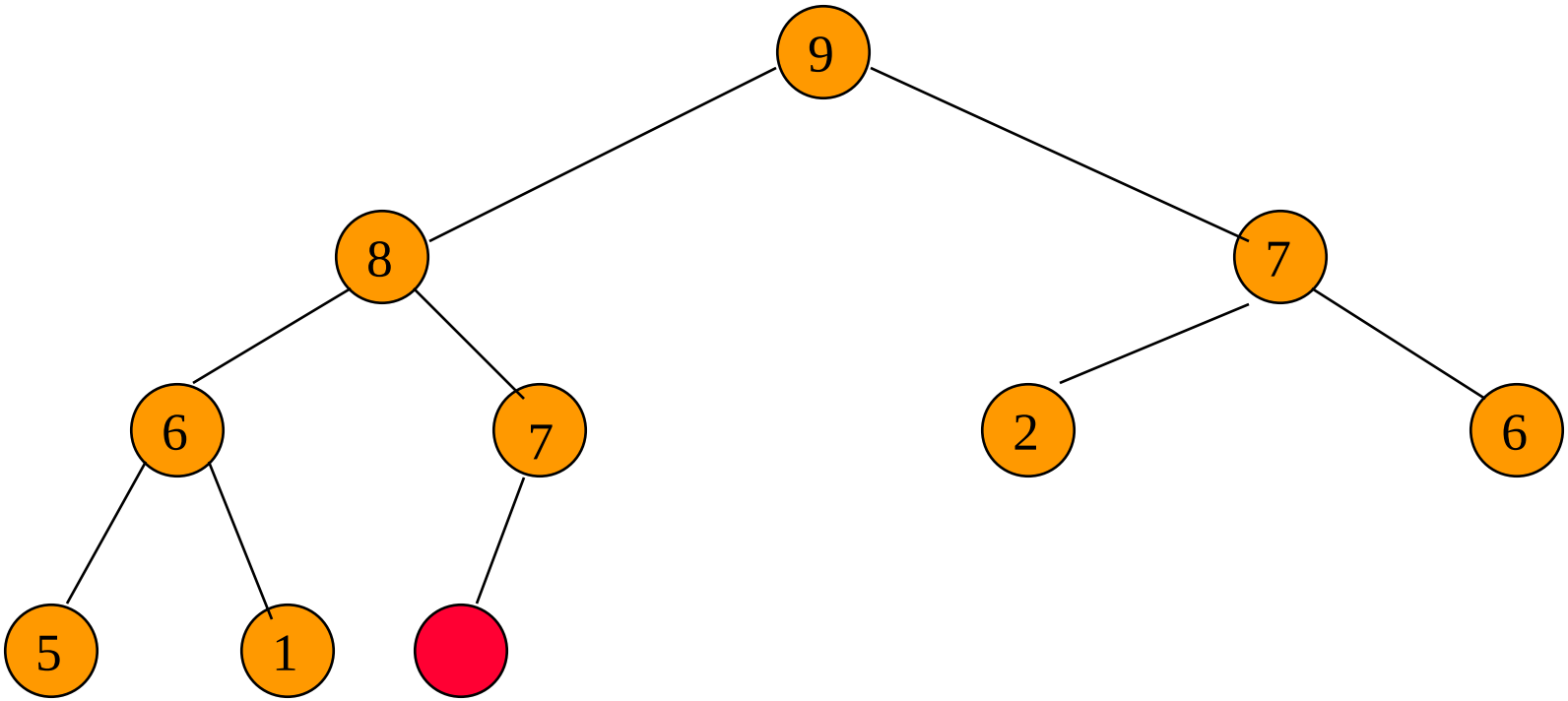
Complete binary tree with 10 nodes.

Putting An Element Into A Max Heap



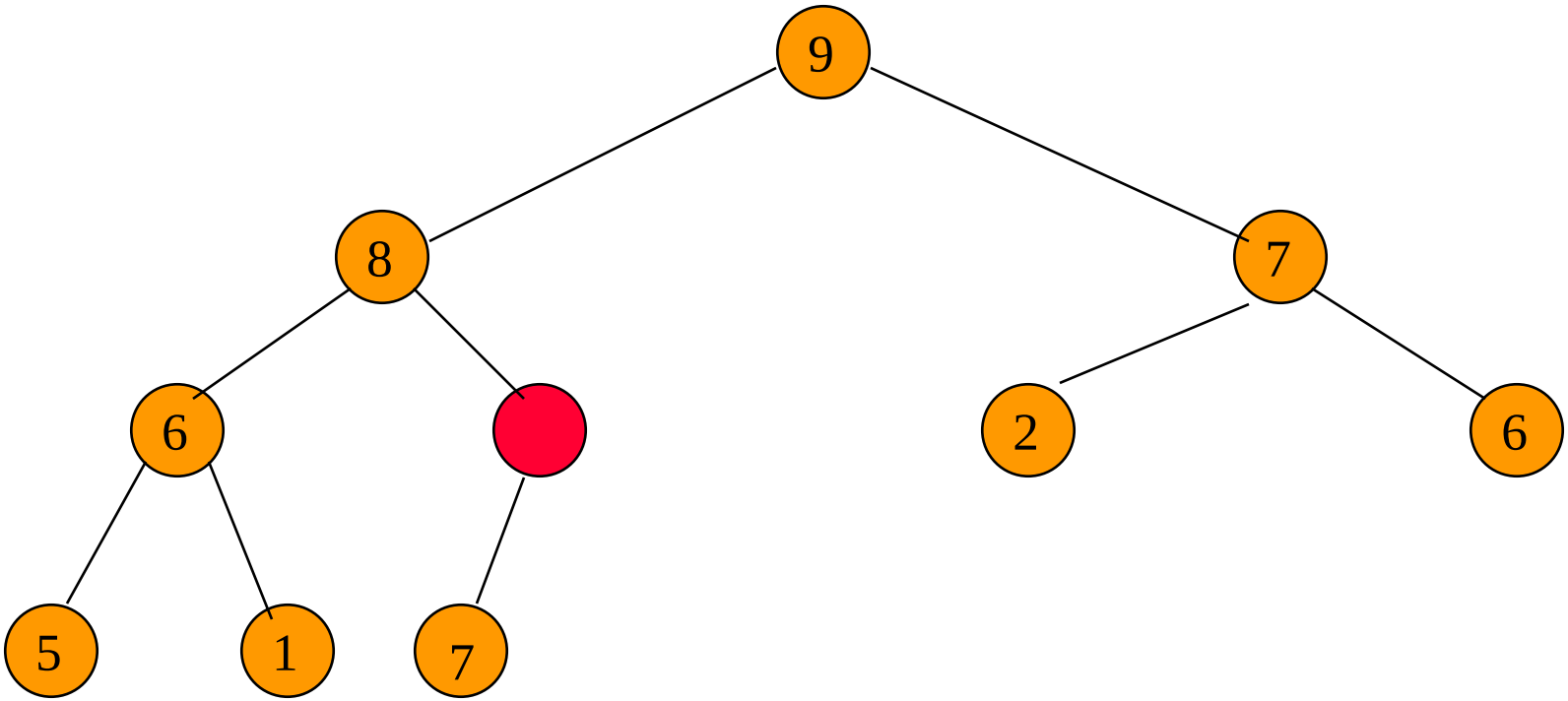
New element is 5.

Putting An Element Into A Max Heap



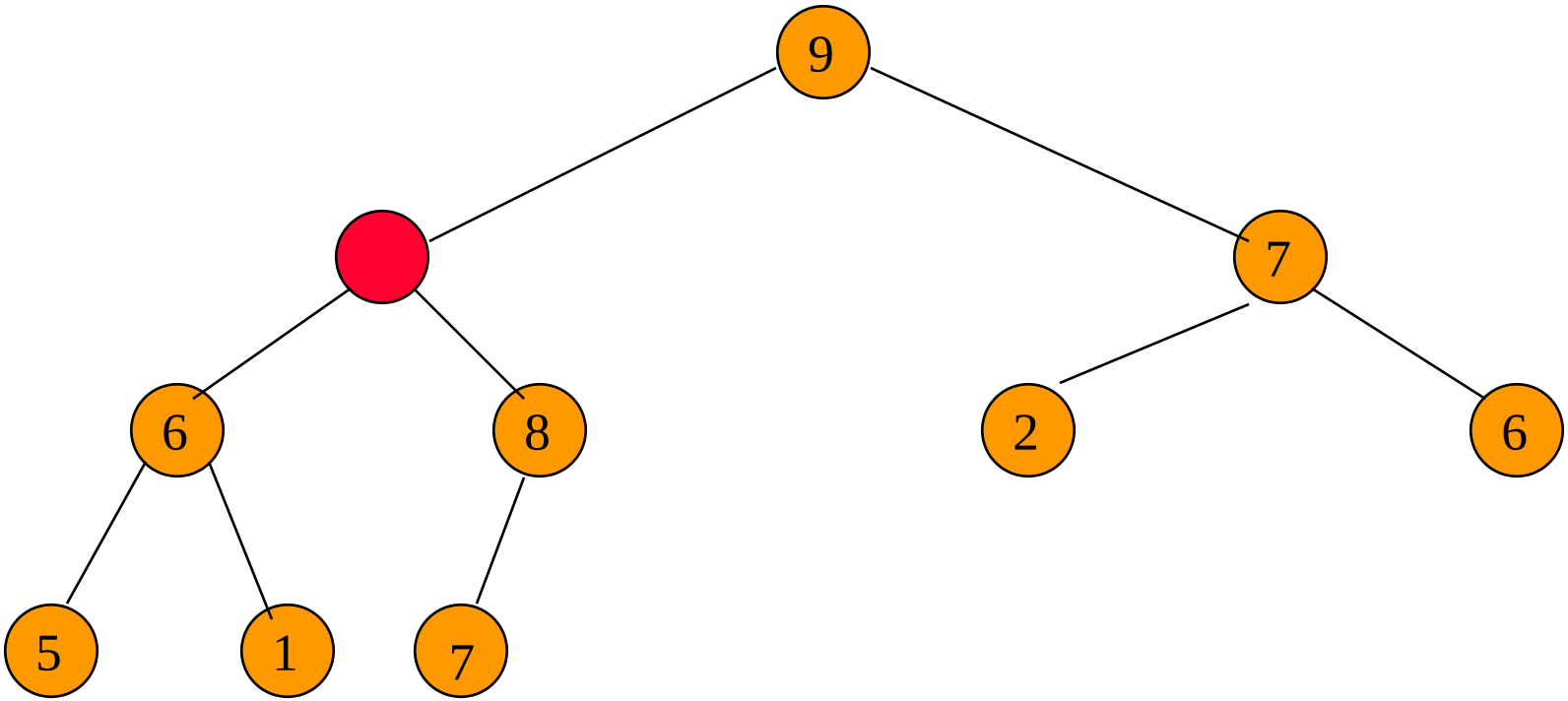
New element is 20.

Putting An Element Into A Max Heap



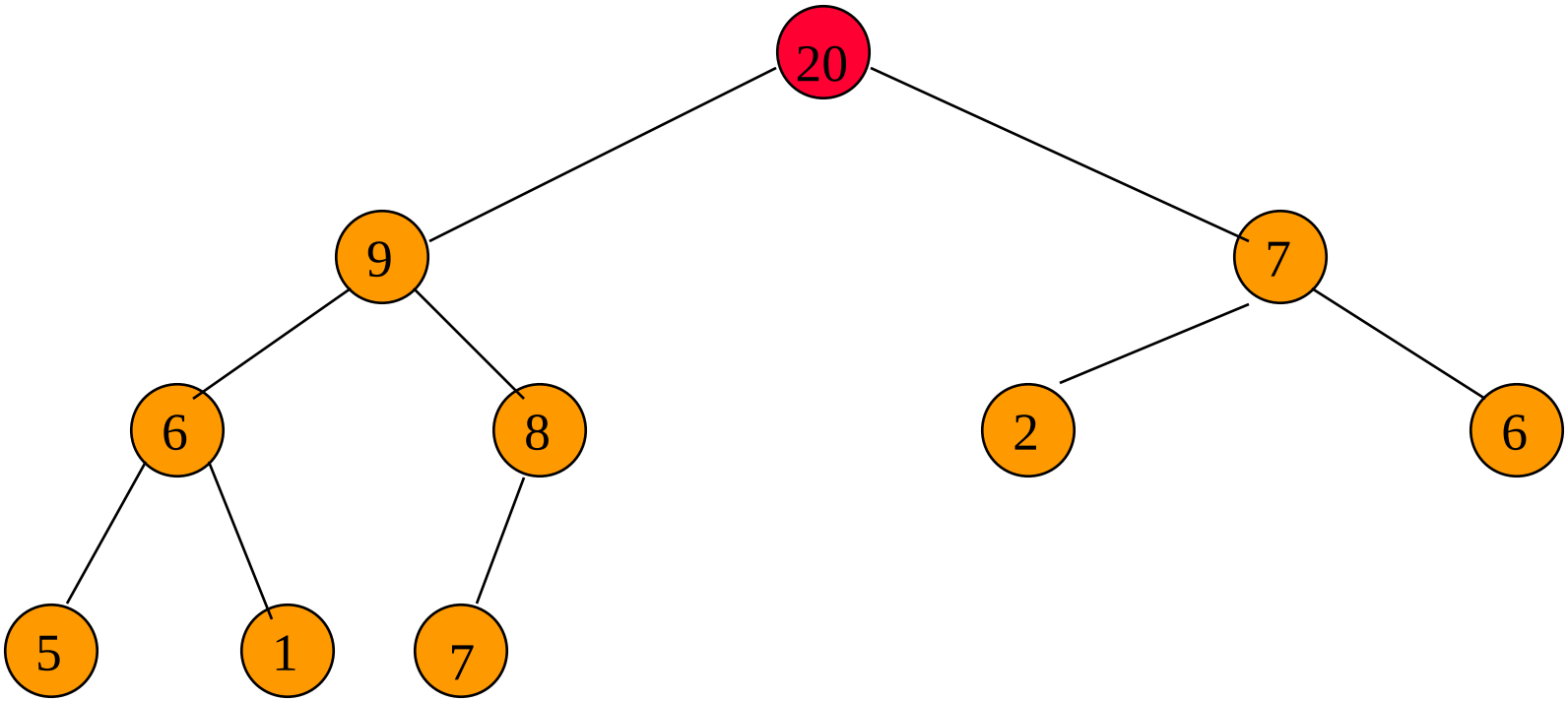
New element is 20.

Putting An Element Into A Max Heap



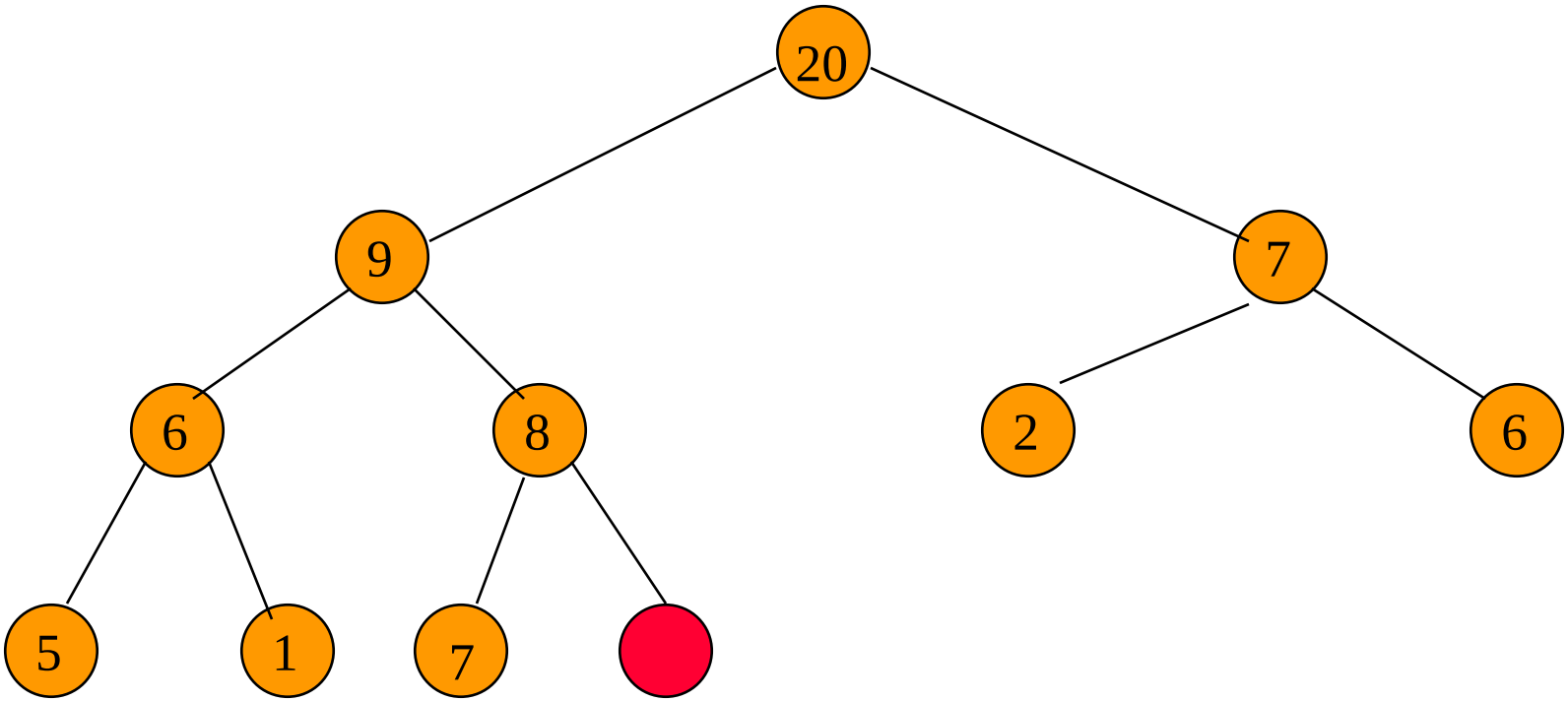
New element is 20.

Putting An Element Into A Max Heap



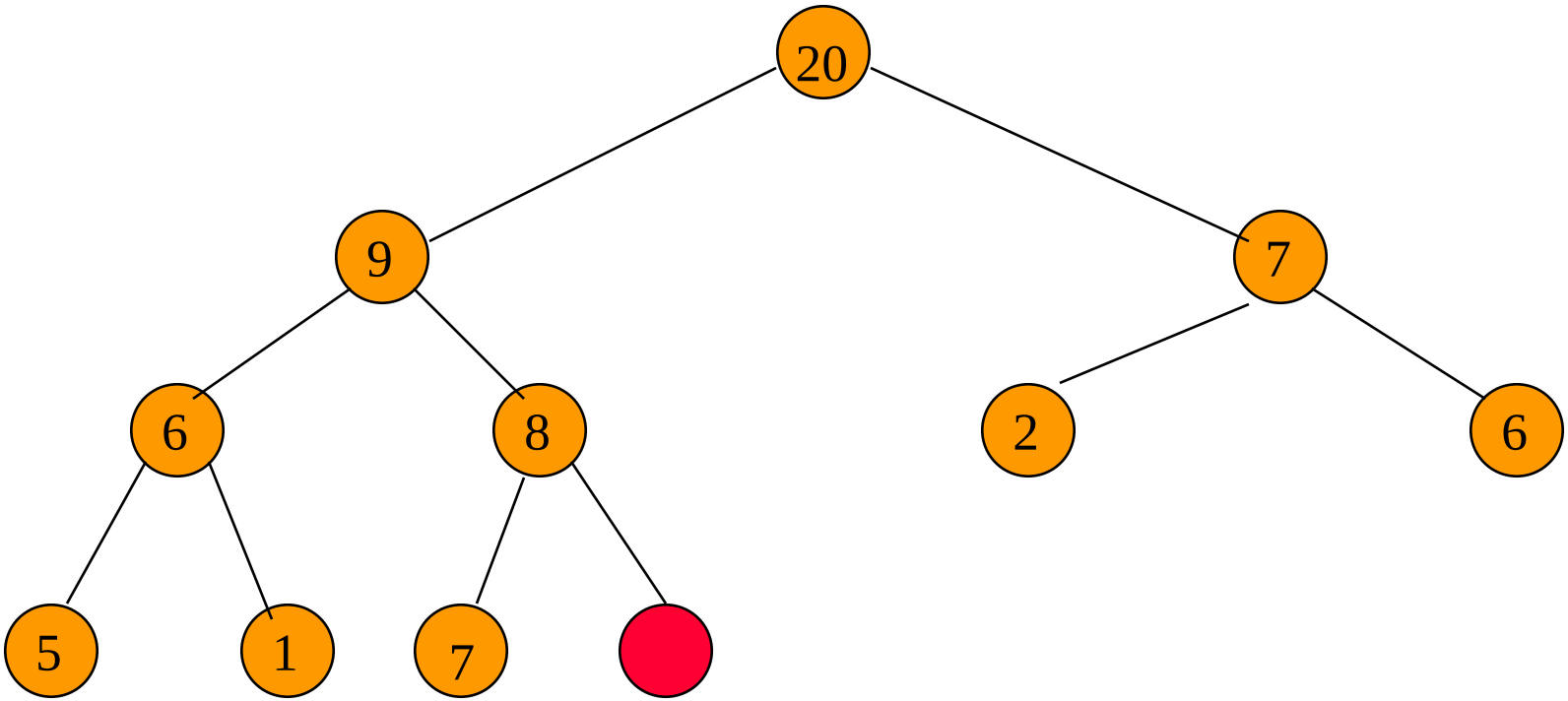
New element is 20.

Putting An Element Into A Max Heap



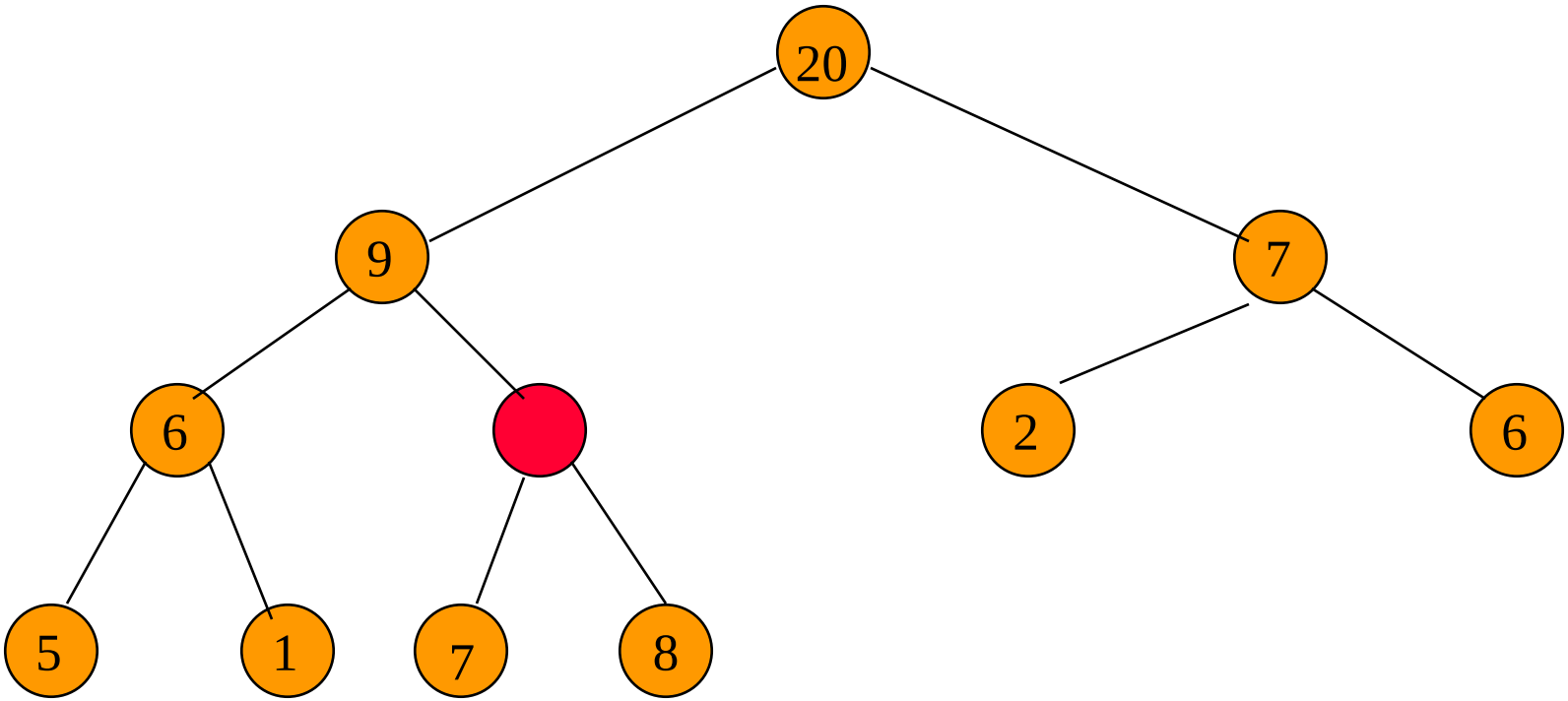
Complete binary tree with 11 nodes.

Putting An Element Into A Max Heap



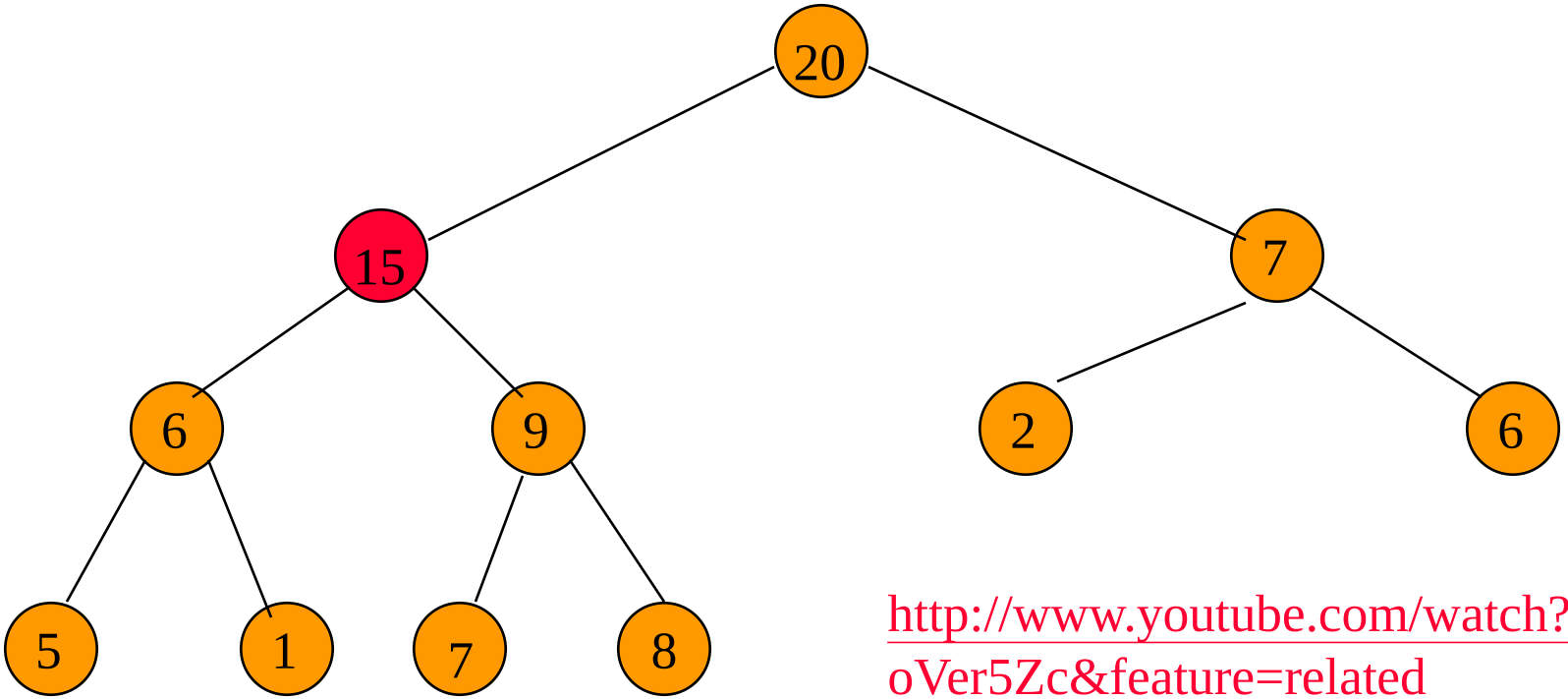
New element is 15.

Putting An Element Into A Max Heap



New element is 15.

Putting An Element Into A Max Heap

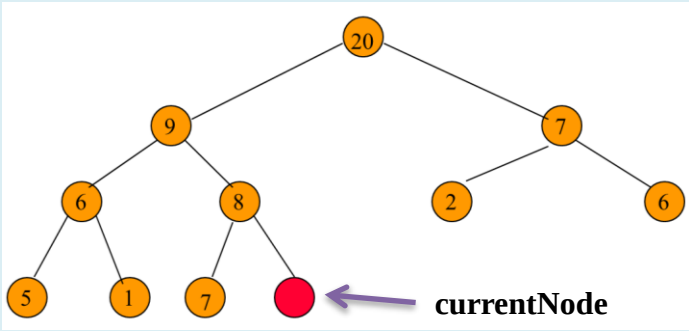


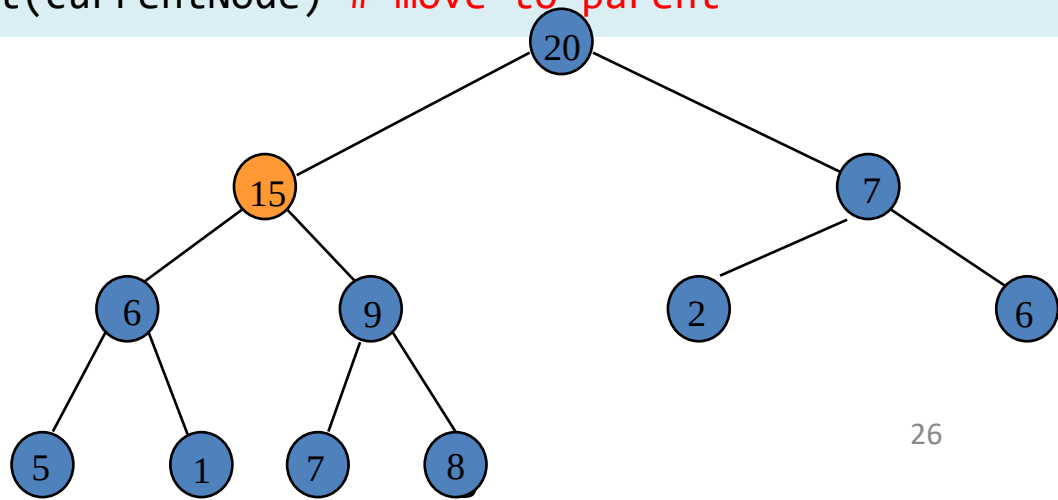
http://www.youtube.com/watch?v=wkWHoVer5Zc&feature=related

New element is 15.

MaxHeap put

```
# Function to put a node into the heap
    def put(self, theElement):

        if self.size >= self.maxsize:
            return
        # find place for theElement
        # currentNode starts at new leaf and moves
        self.size += 1
        self.Heap[self.size] = theElement

        currentNode = self.size

        while (self.Heap[currentNode] > self.Heap[self.parent(currentNode)]):
            # cannot put theElement in Heap[currentNode]
            self.swap(currentNode, self.parent(currentNode)) # move element down
            currentNode = self.parent(currentNode) # move to parent
```

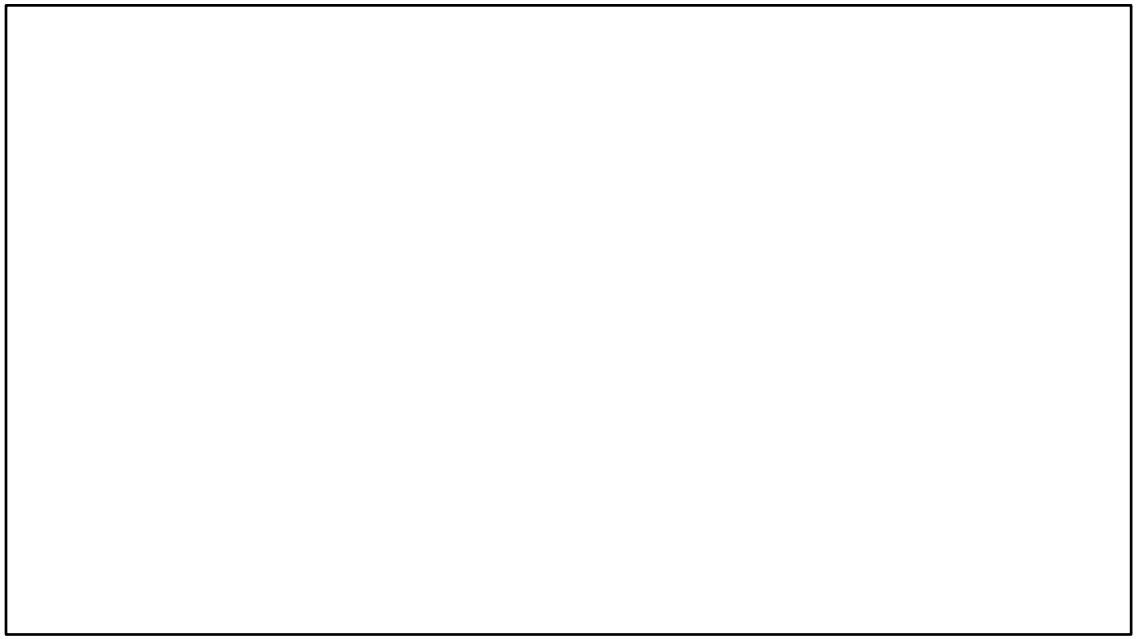




demo: MaxHeap.py

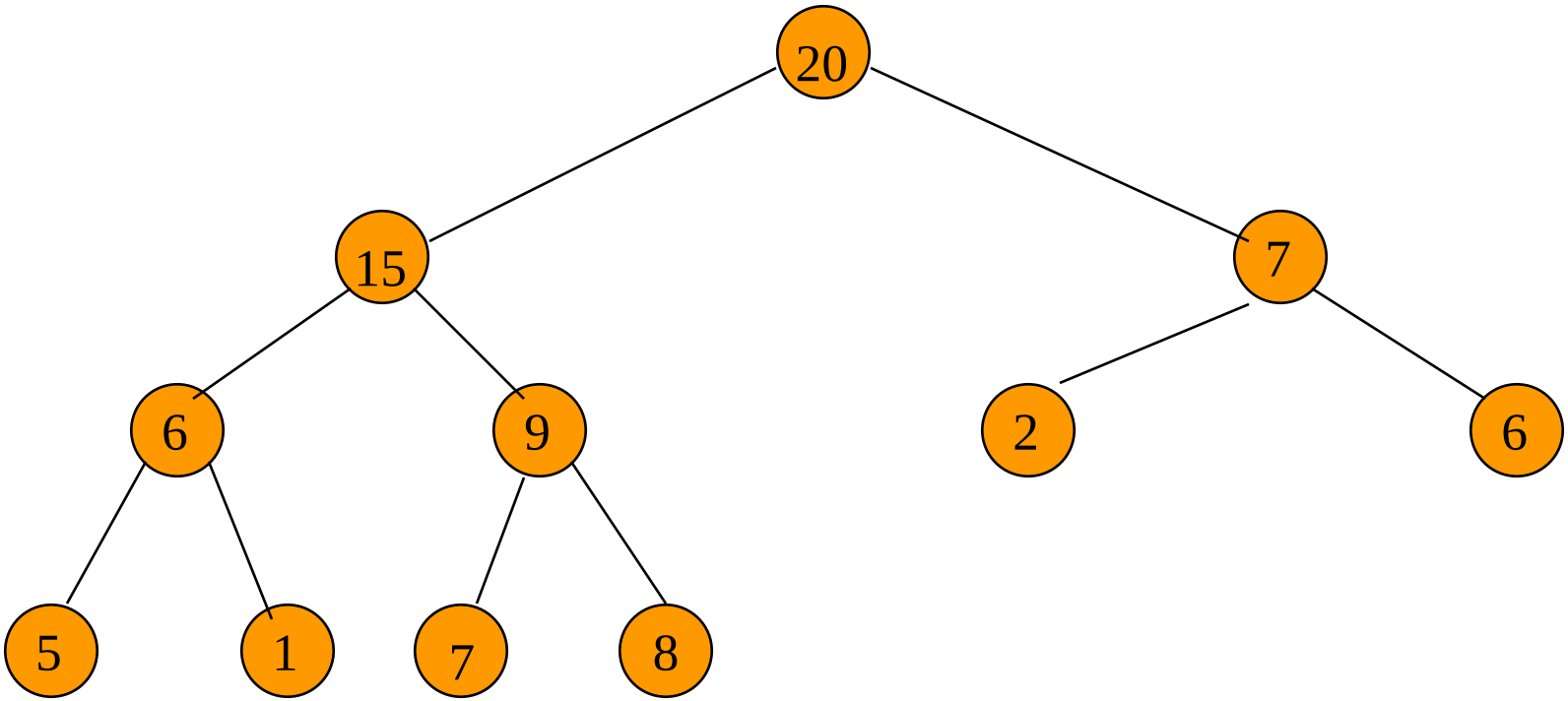
Exercise

- Show the heap (tree) you will have after inserting the following values:
- 80, 40, 30, 60, 81, 90, 100, 10



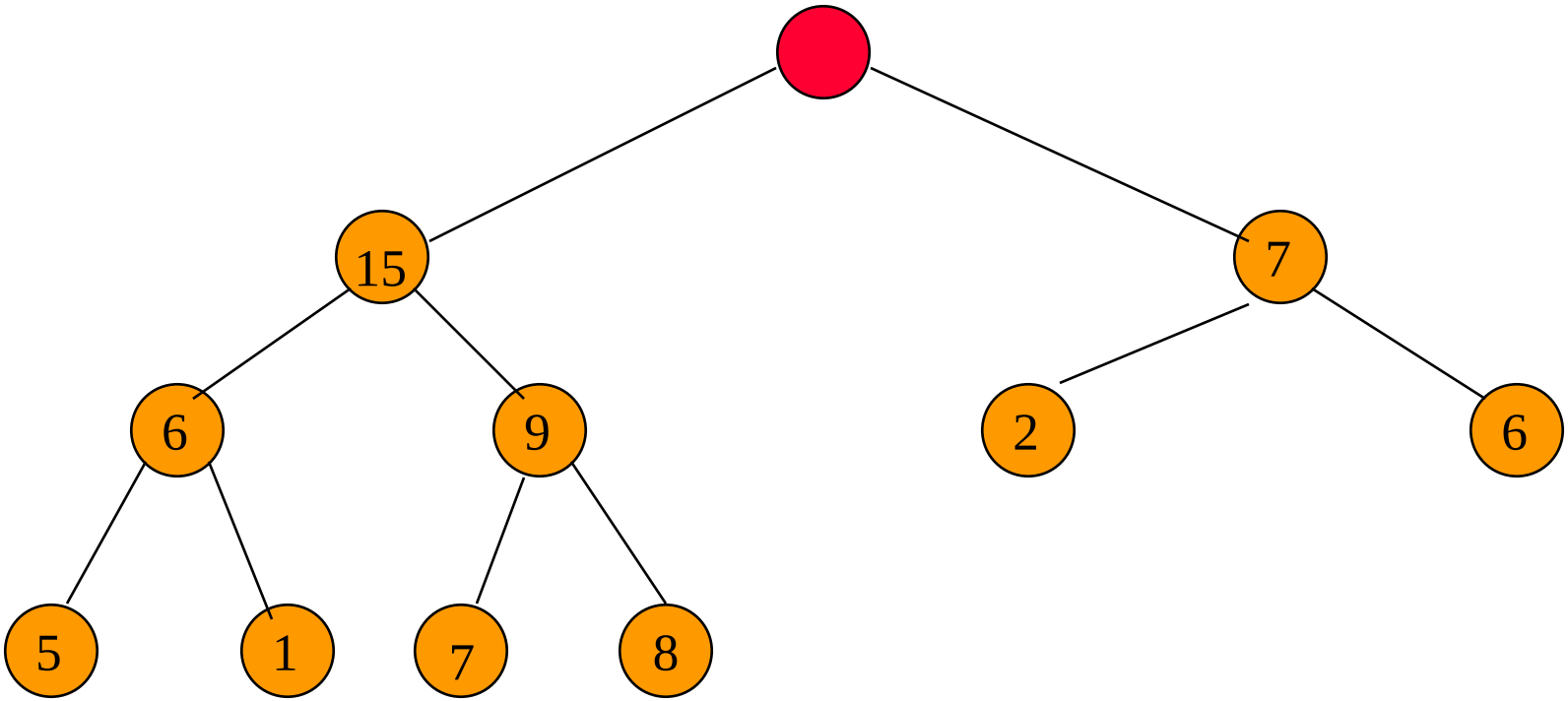
Implementation of Heap - Remove

Removing The Max Element



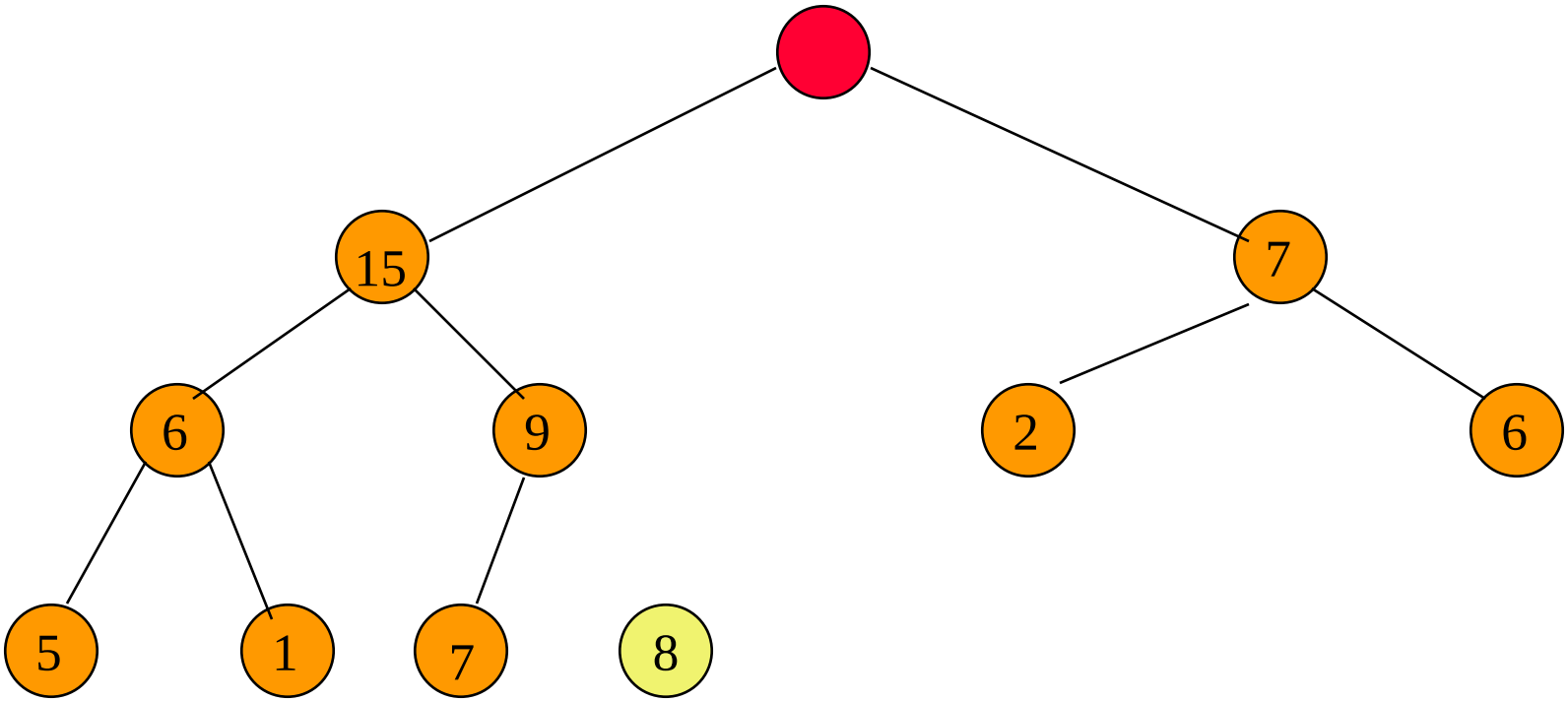
Max element is in the root.

Removing The Max Element



After max element is removed.

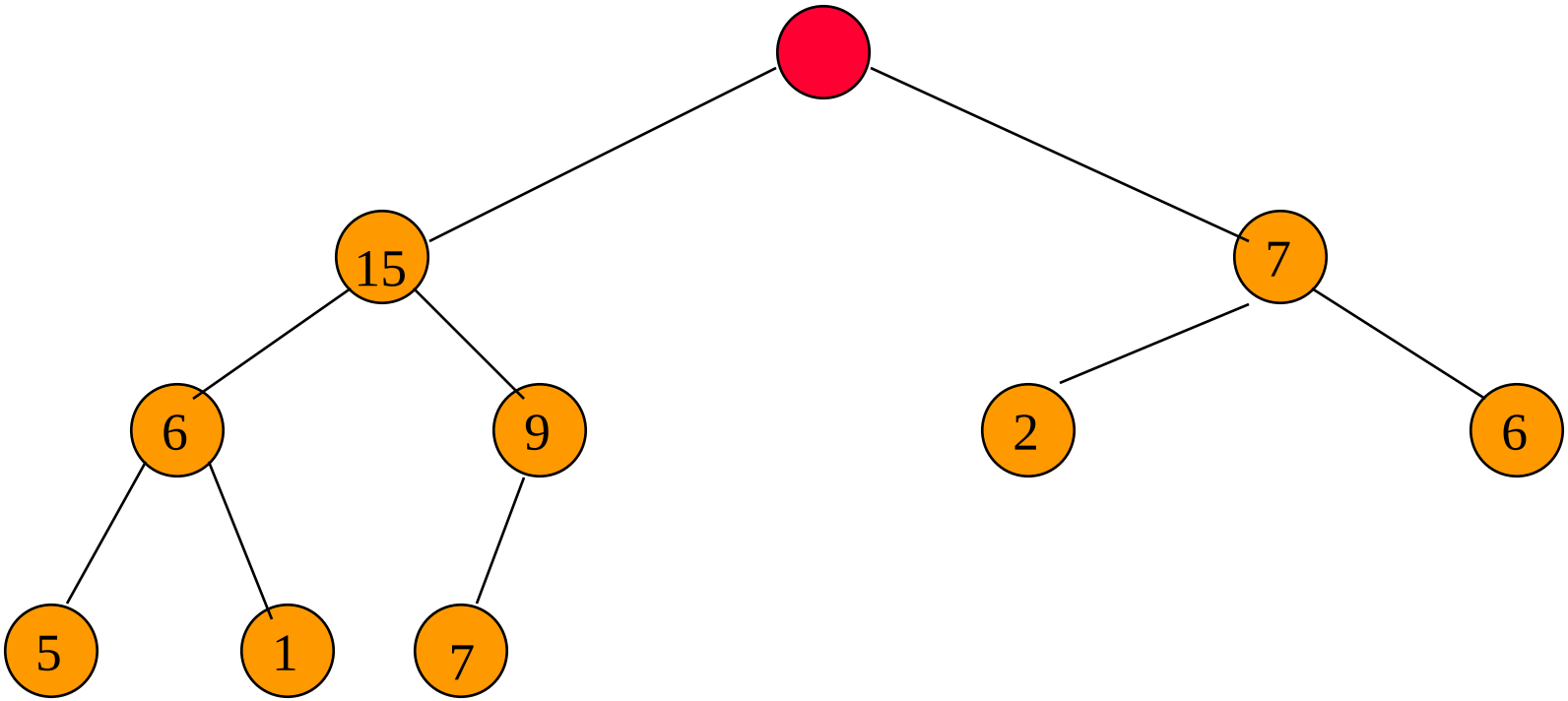
Removing The Max Element



Heap with 10 nodes.

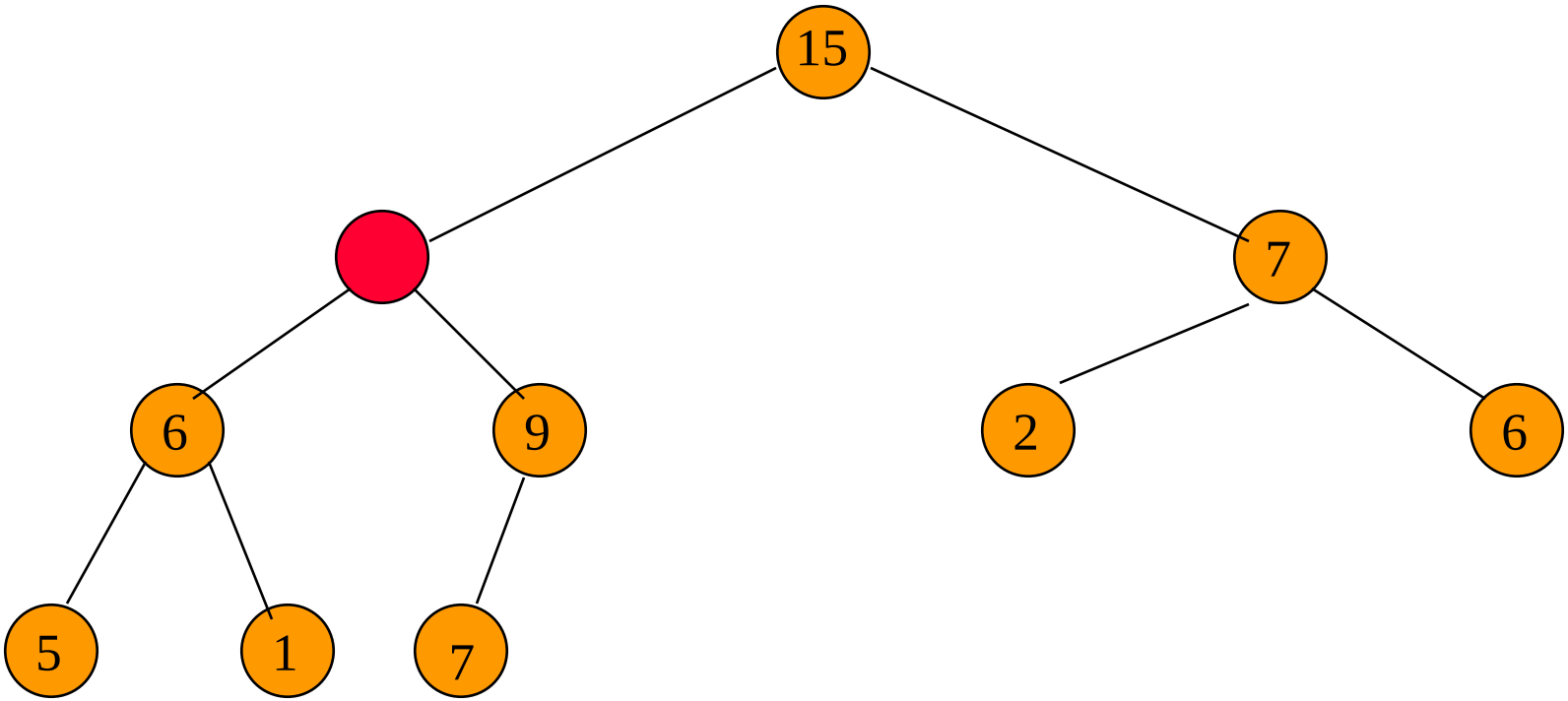
Reinsert 8 into the heap. (at the root)

Removing The Max Element



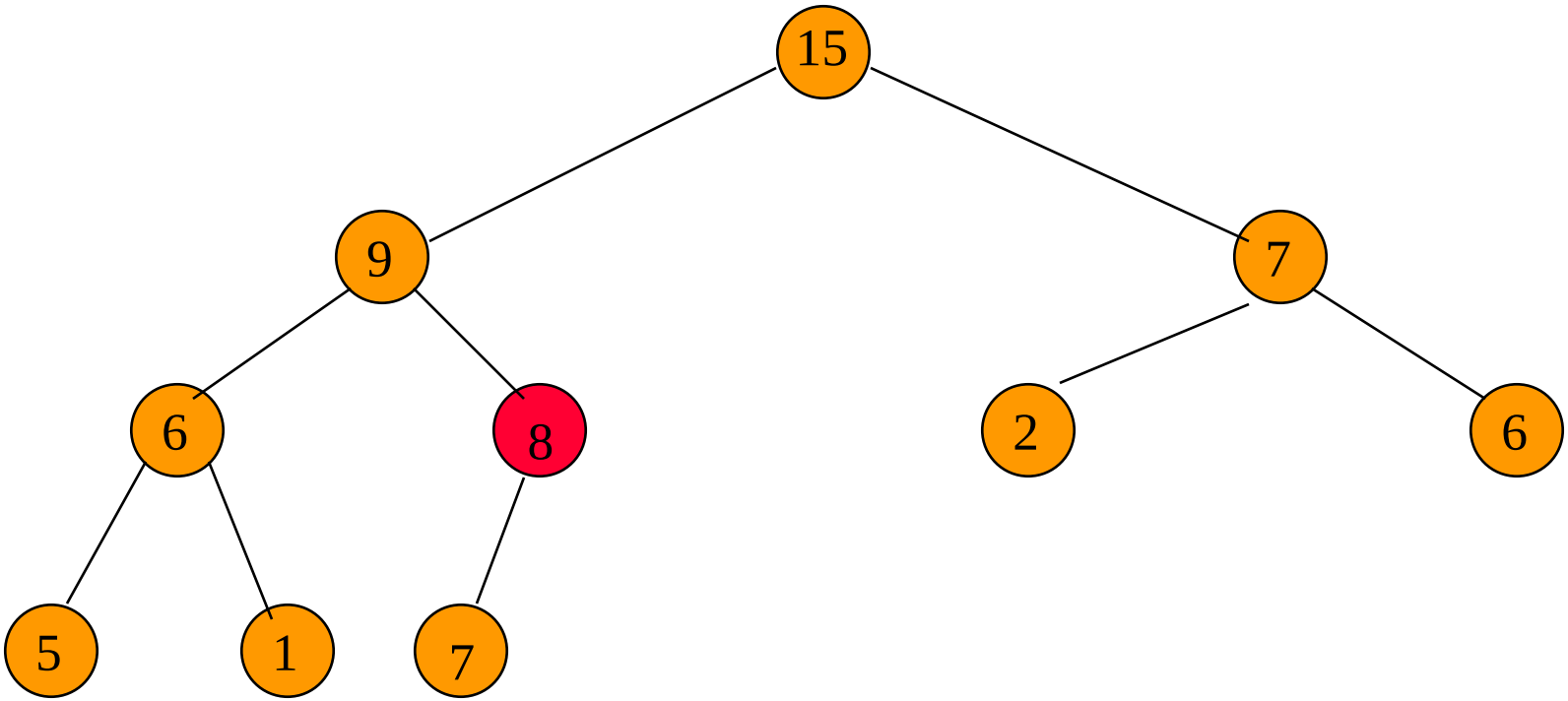
Reinsert 8 into the heap.

Removing The Max Element



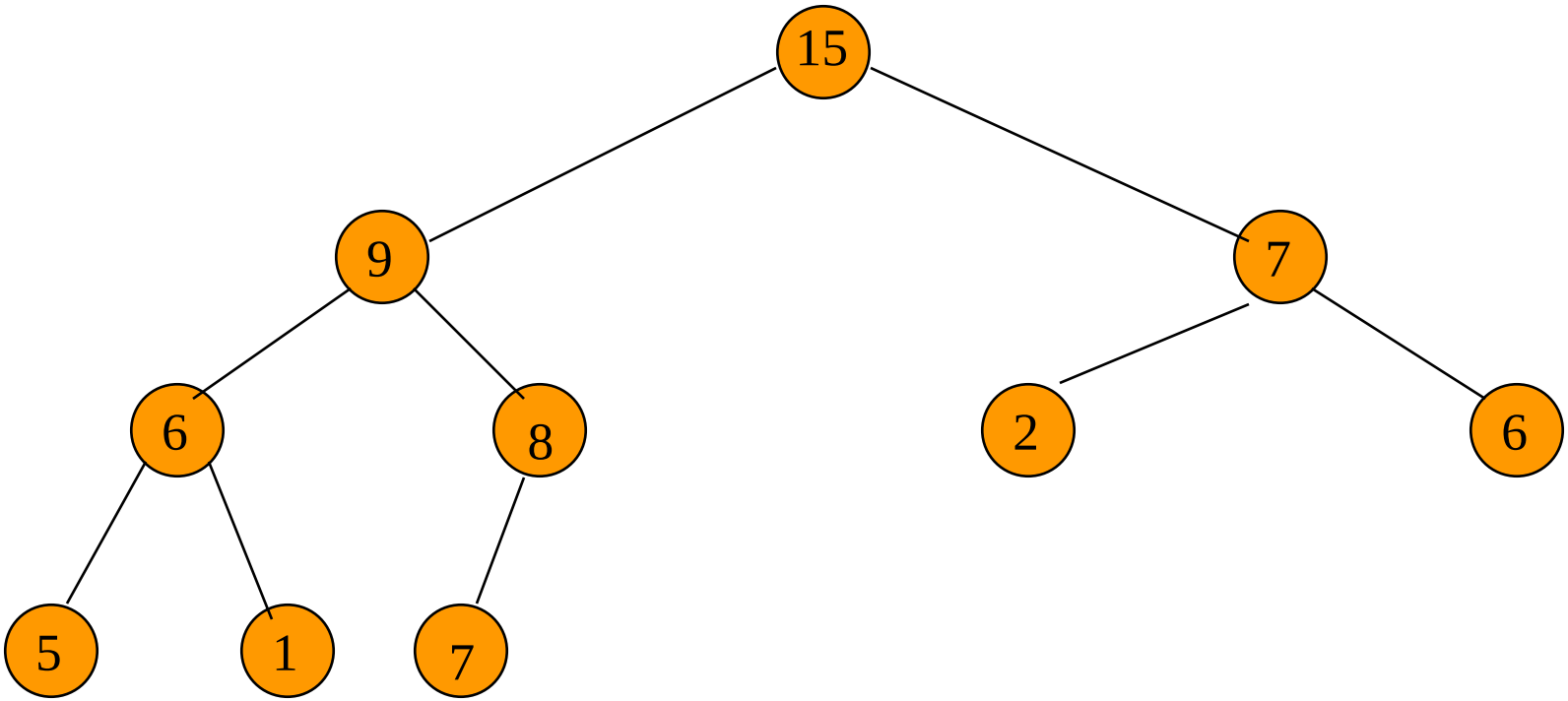
Reinsert 8 into the heap.

Removing The Max Element



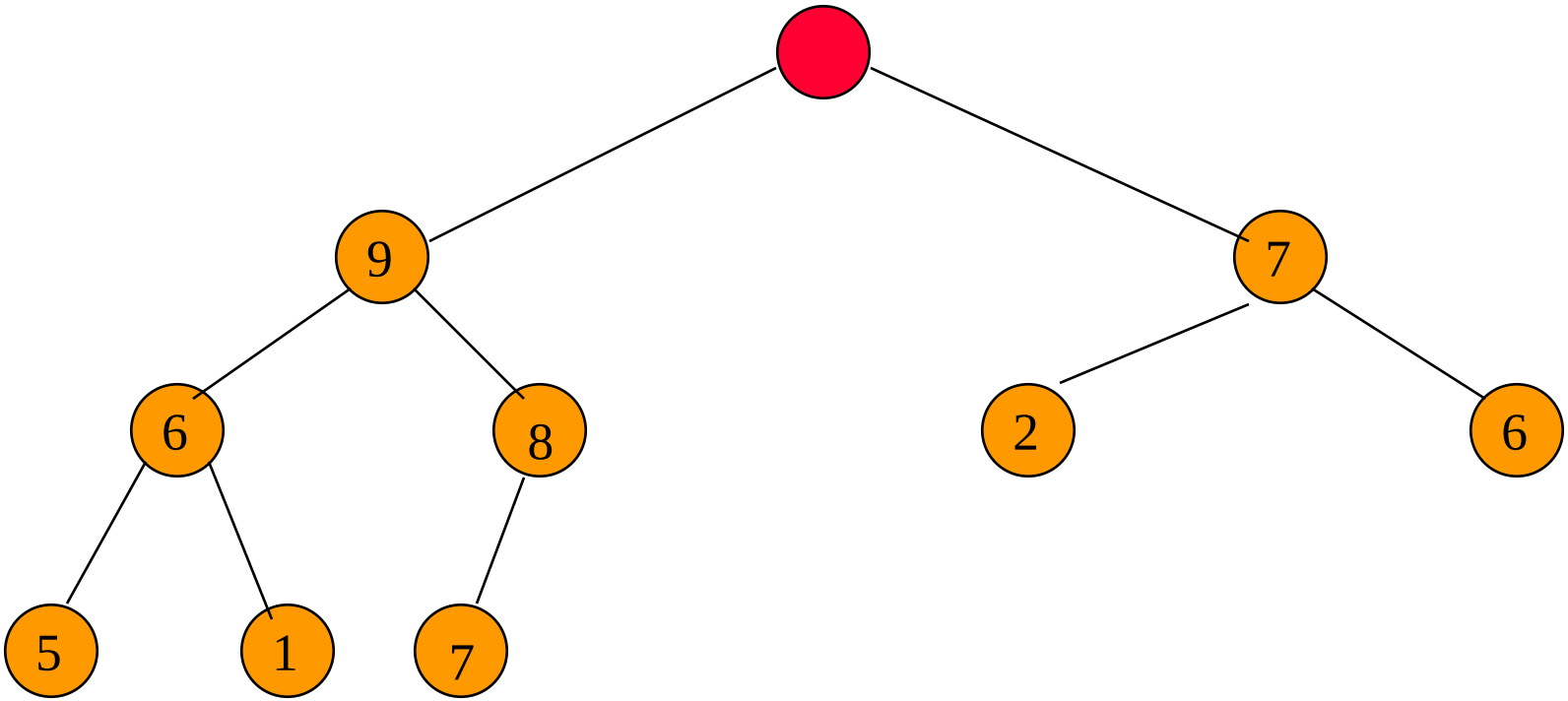
Reinsert 8 into the heap.

Removing The Max Element



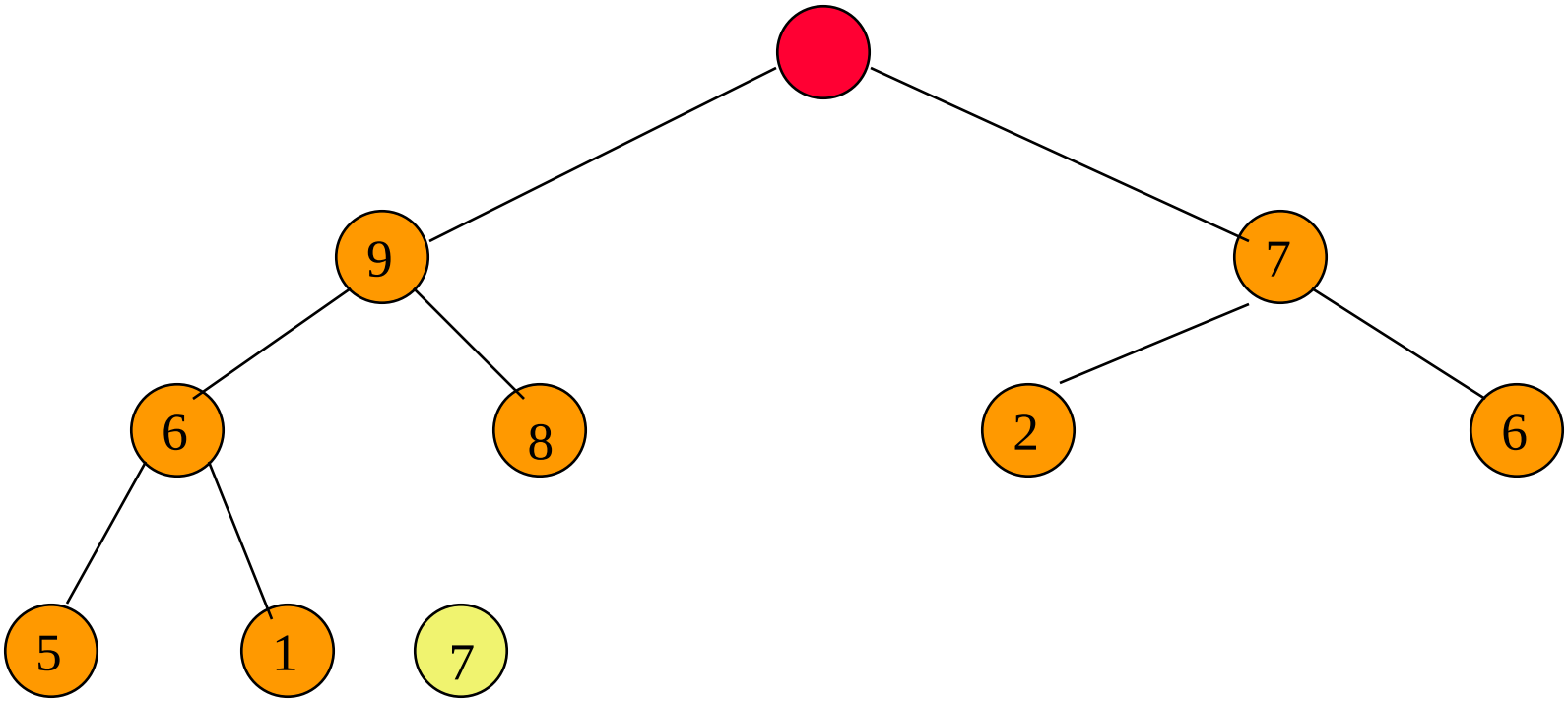
Max element is 15.

Removing The Max Element



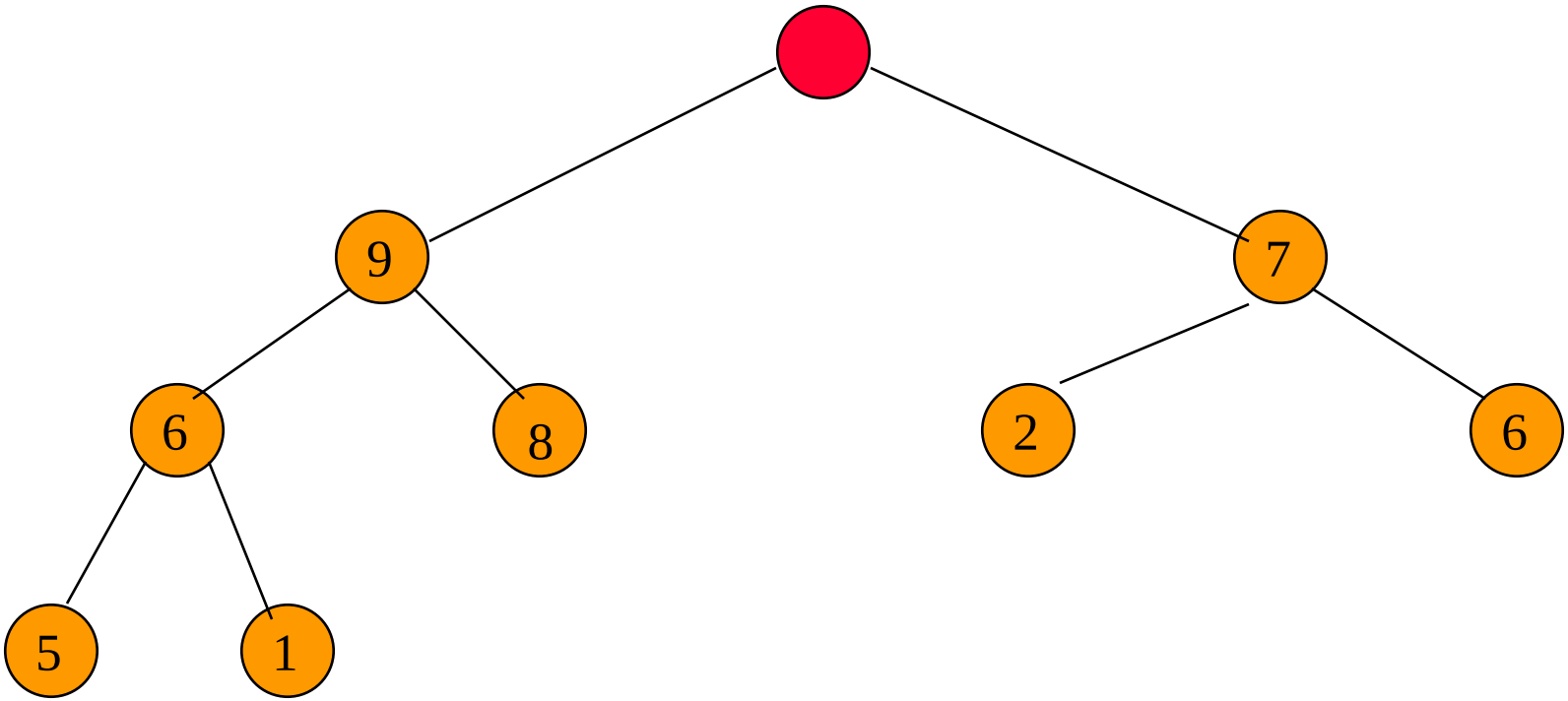
After max element is removed.

Removing The Max Element



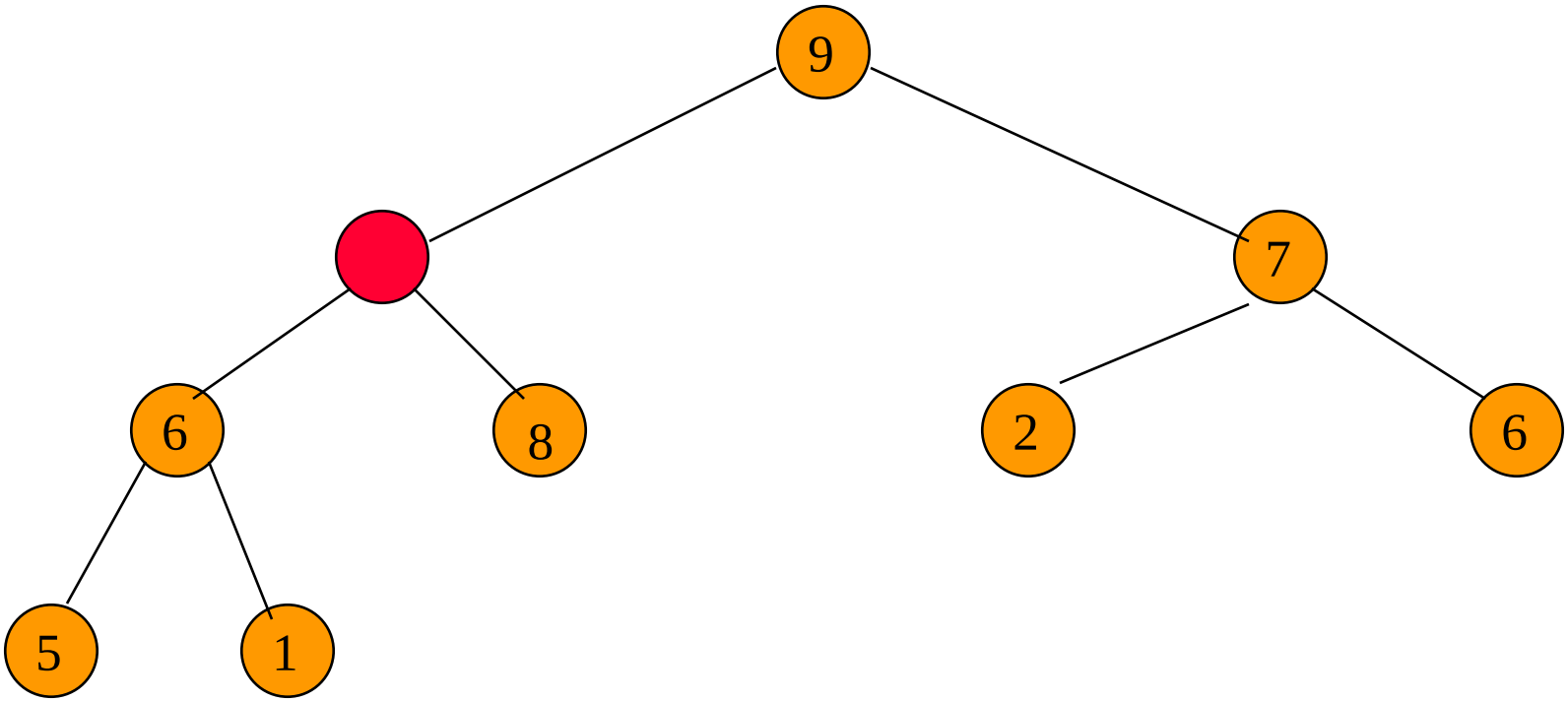
Heap with 9 nodes.

Removing The Max Element



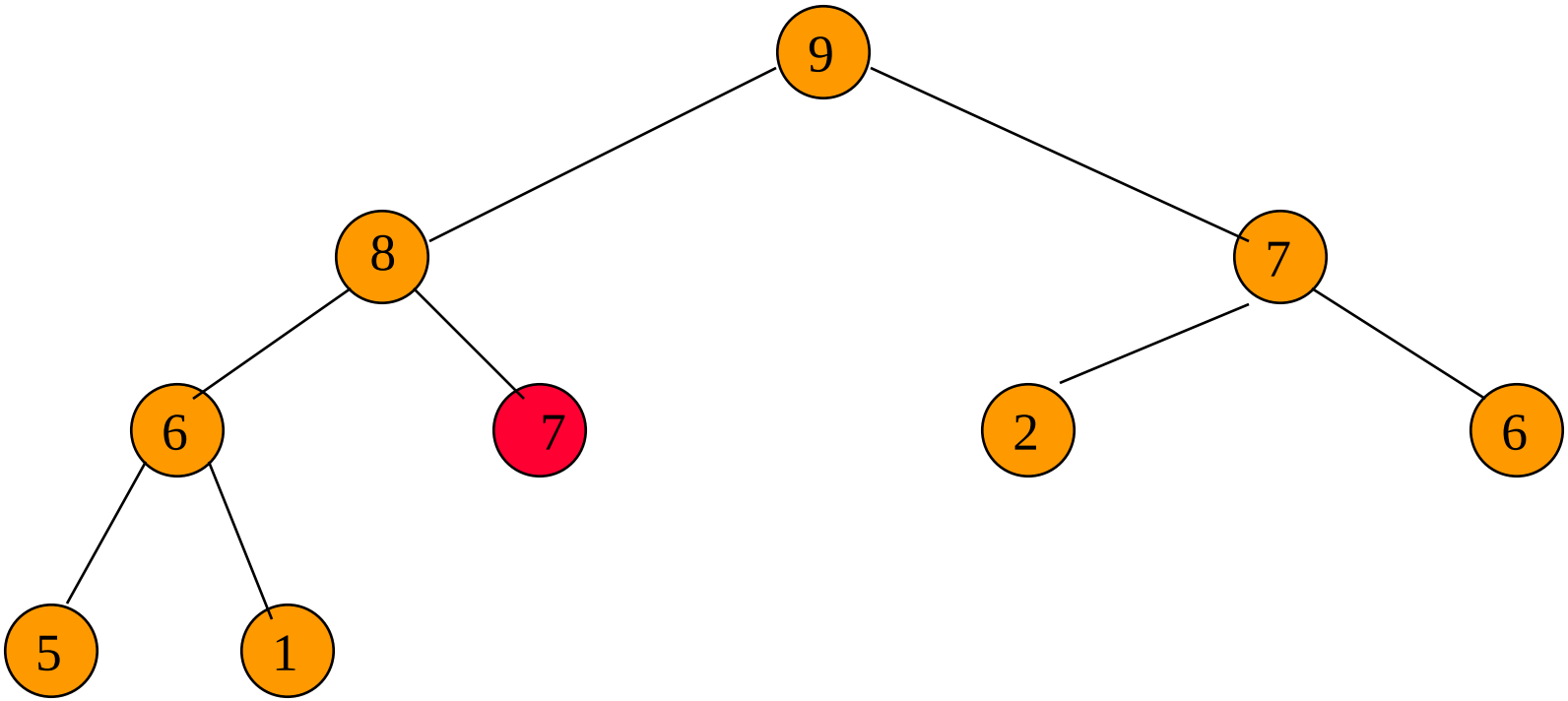
Reinsert 7.

Removing The Max Element



Reinsert 7.

Removing The Max Element

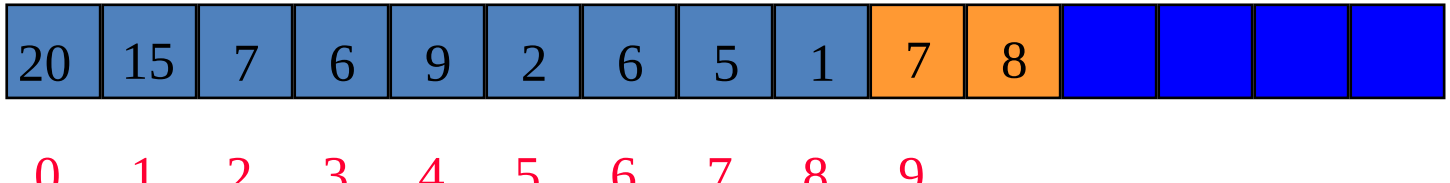


Reinsert 7.

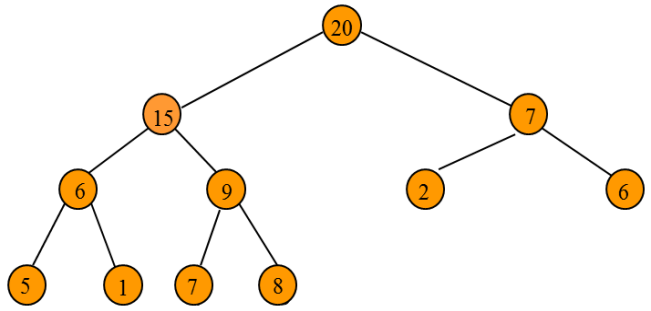
MaxHeap remove

```
# Function to remove and return the maximum element from the heap
    def removeMax(self):

        popped = self.Heap[self.FRONT]  # max element
        # reheapify
        self.Heap[self.FRONT] = self.Heap[self.size]
        self.size -= 1  # decrease the size
        self.maxHeapify(self.FRONT)

        return popped
```

heapify() method is shown in the coming slides



MaxHeap heapify




```
# Function to heapify the node at pos
    def maxHeapify(self, pos):
        # If the node is a non-leaf node and smaller
        # than any of its child
        if not self.isLeaf(pos):
            if (self.Heap[pos] < self.Heap[self.leftChild(pos)] or
                self.Heap[pos] < self.Heap[self.rightChild(pos)]):

                # Swap with the left child and heapify the left child
                if (self.Heap[self.leftChild(pos)] >
                    self.Heap[self.rightChild(pos)]):
                    self.swap(pos, self.leftChild(pos))
                    self.maxHeapify(self.leftChild(pos))

                # Swap with the right child and heapify the right child
                else:
                    self.swap(pos, self.rightChild(pos))
                    self.maxHeapify(self.rightChild(pos))
```

MaxHeap

to simplify the implementation, index start from 1

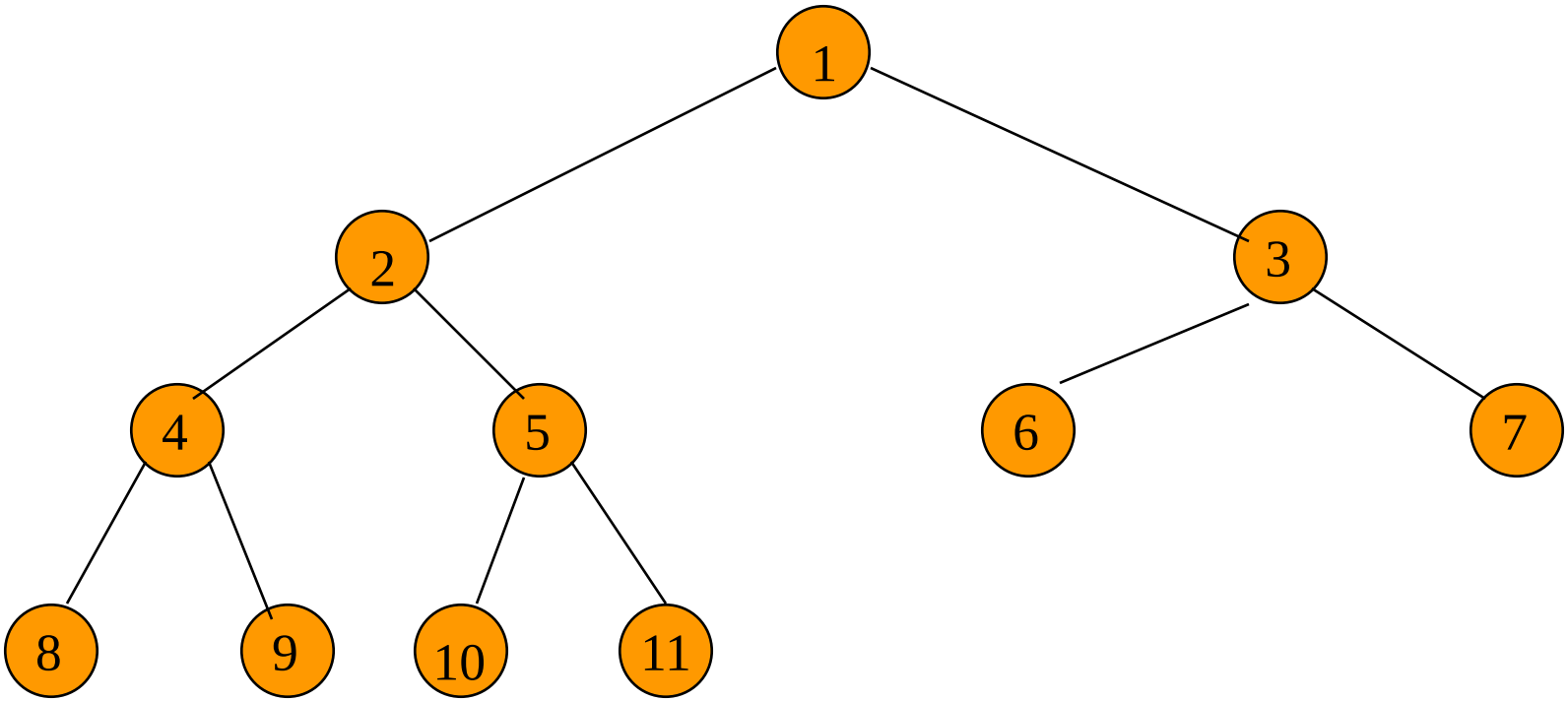
```
def __init__(self, maxsize):

        self.maxsize = maxsize
        self.size = 0
        self.Heap = [0] * (self.maxsize + 1)
        self.Heap[0] = sys.maxsize
        self.FRONT = 1
        self.printed = False
```

demo: MaxHeap.py

Initializing

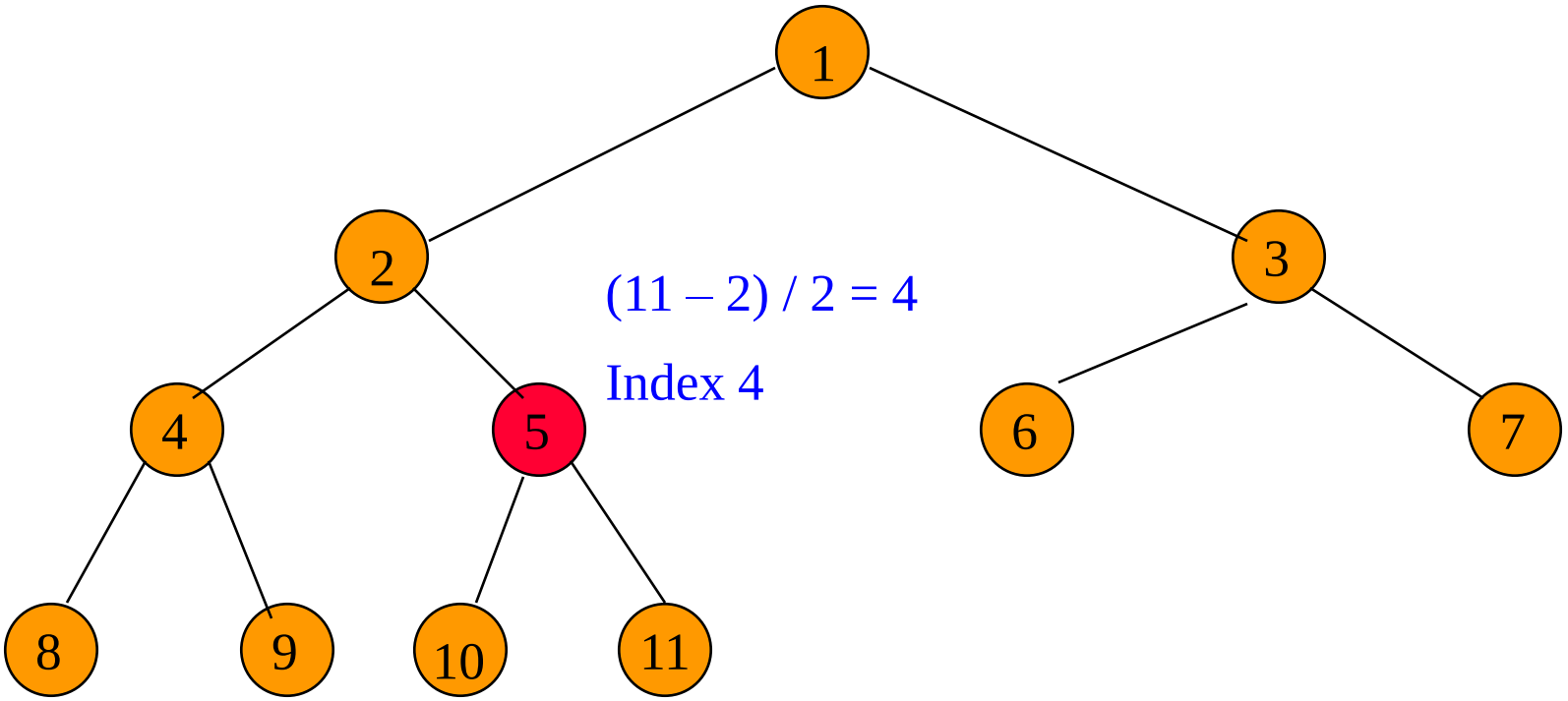
44

Initializing A Max Heap



input array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Initializing A Max Heap

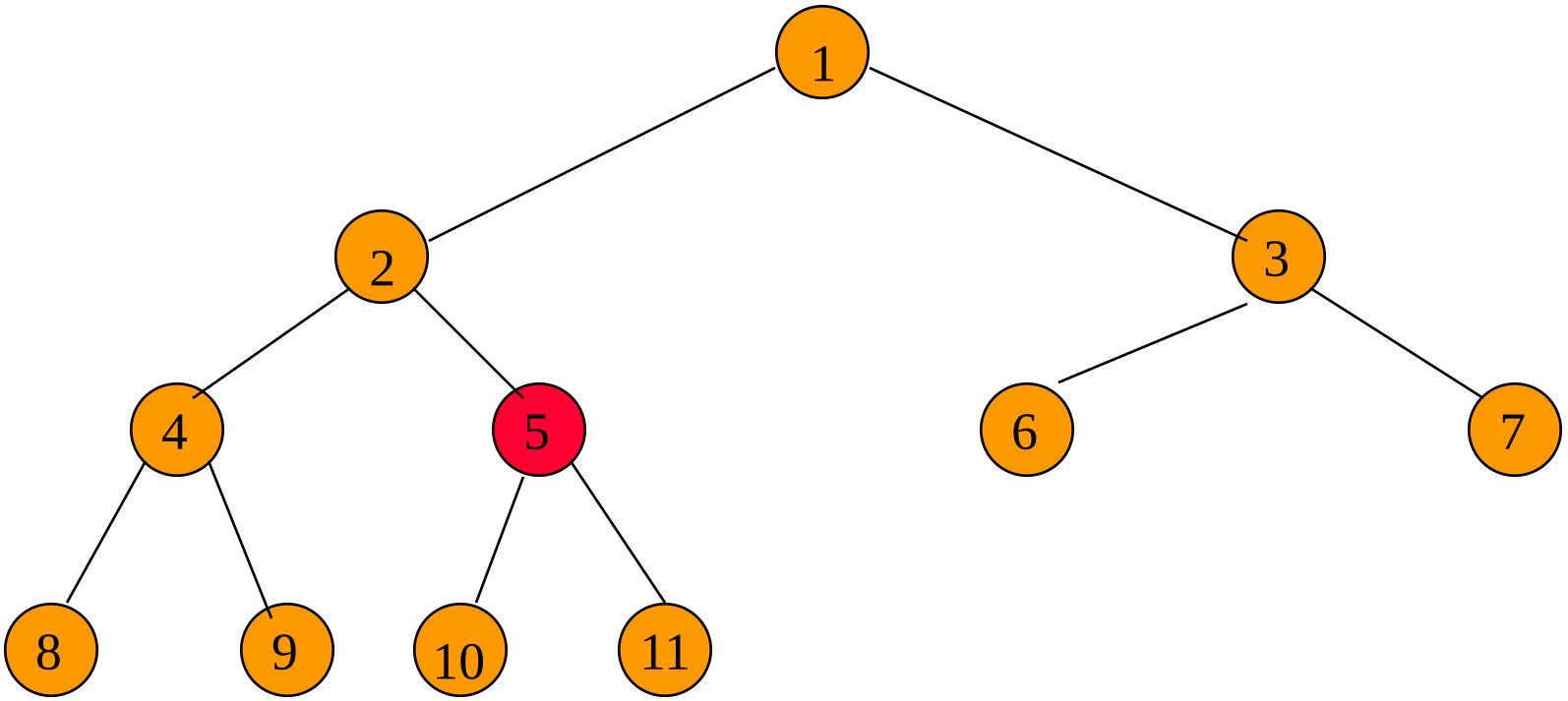


input array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Start at rightmost array position that has a child.

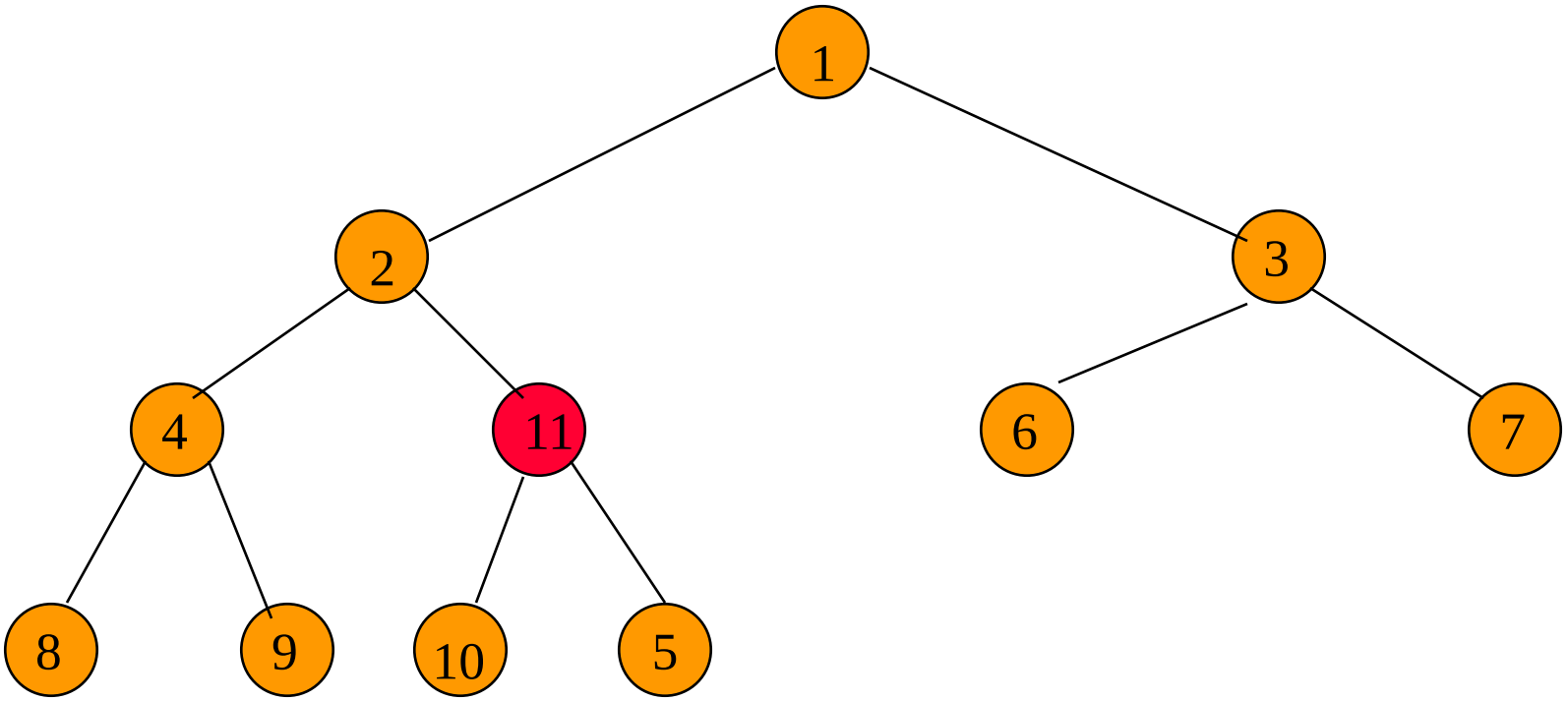
Index is (n-2)/2.

Initializing A Max Heap



input array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

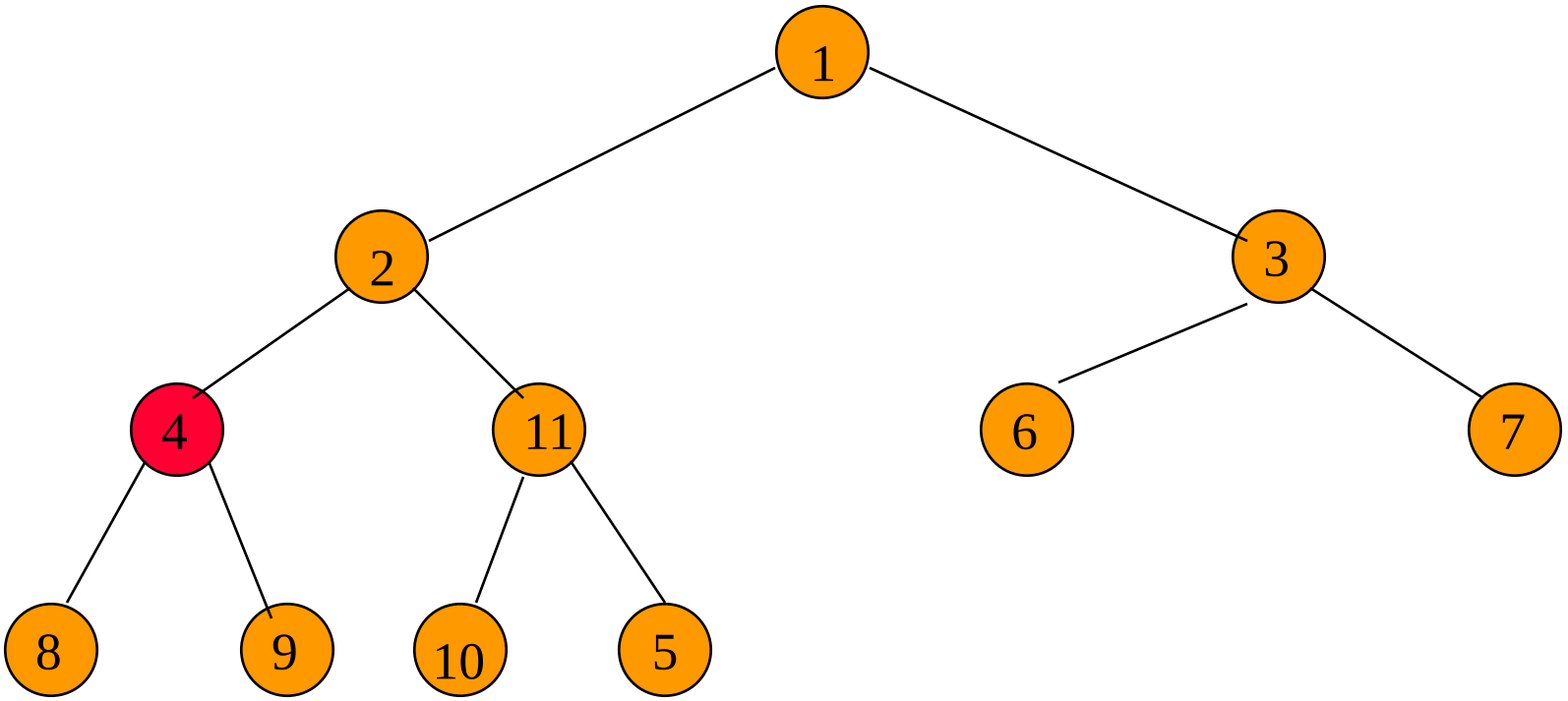
Initializing A Max Heap



input array = [1, 2, 3, 4, 11, 6, 7, 8, 9, 10, 5]

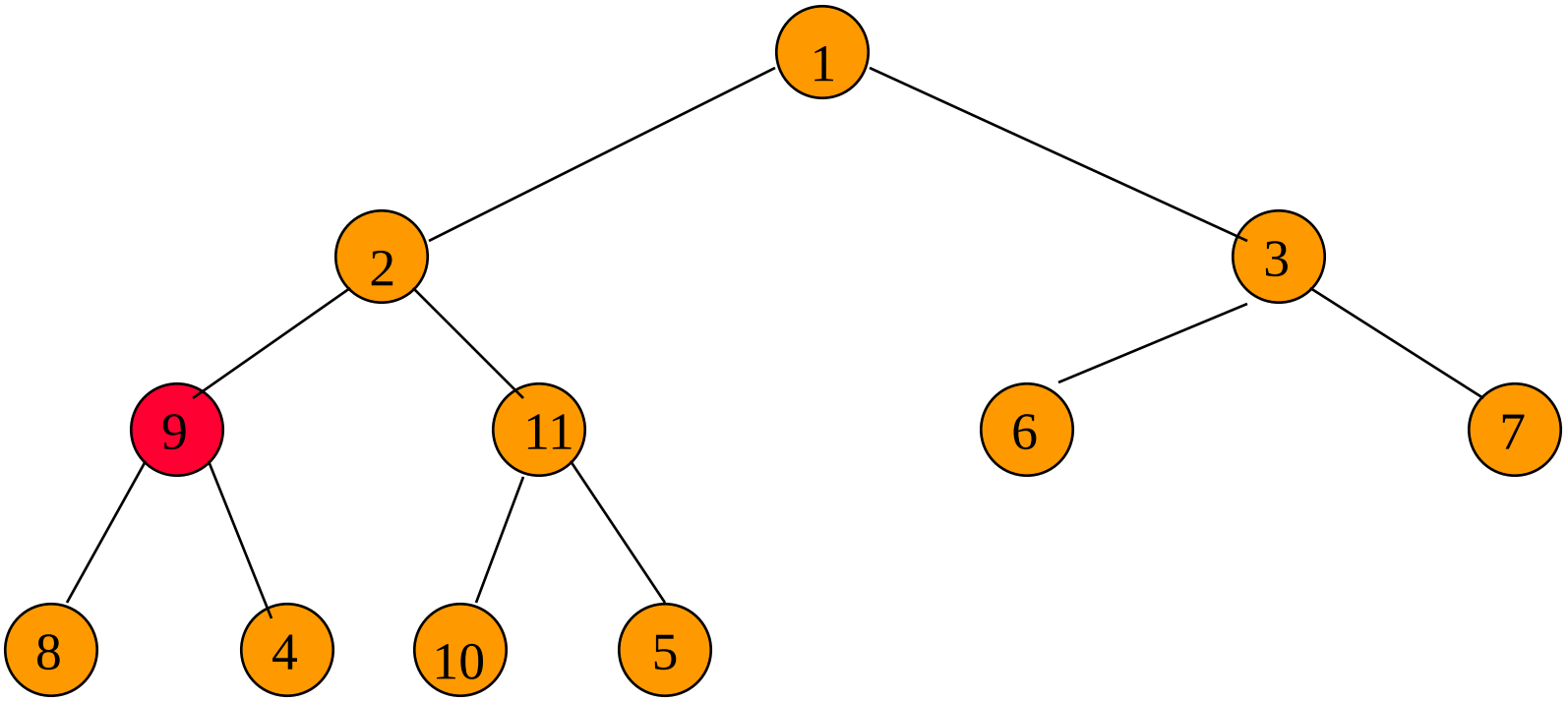
Move to next lower array position.

Initializing A Max Heap



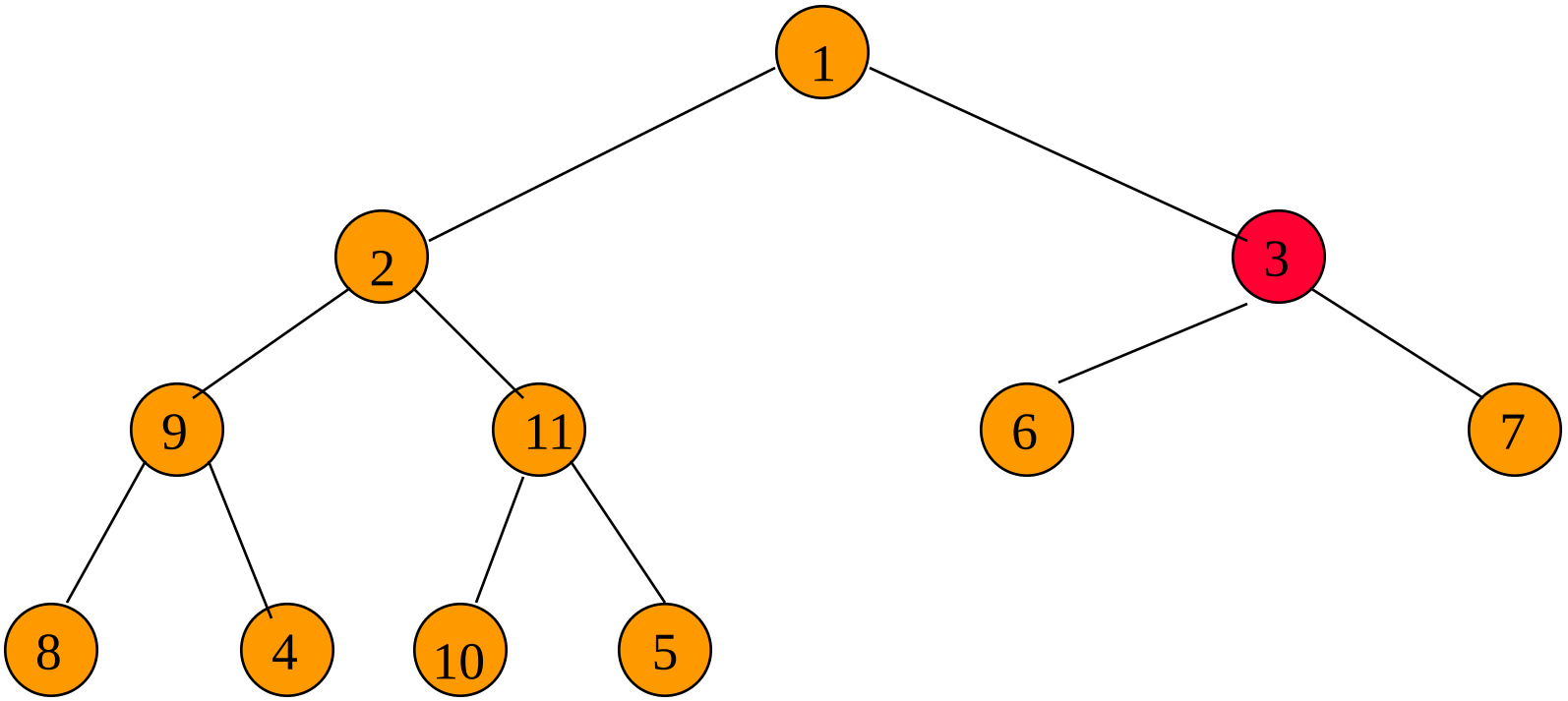
input array = [1, 2, 3, 4, 11, 6, 7, 8, 9, 10, 5]

Initializing A Max Heap



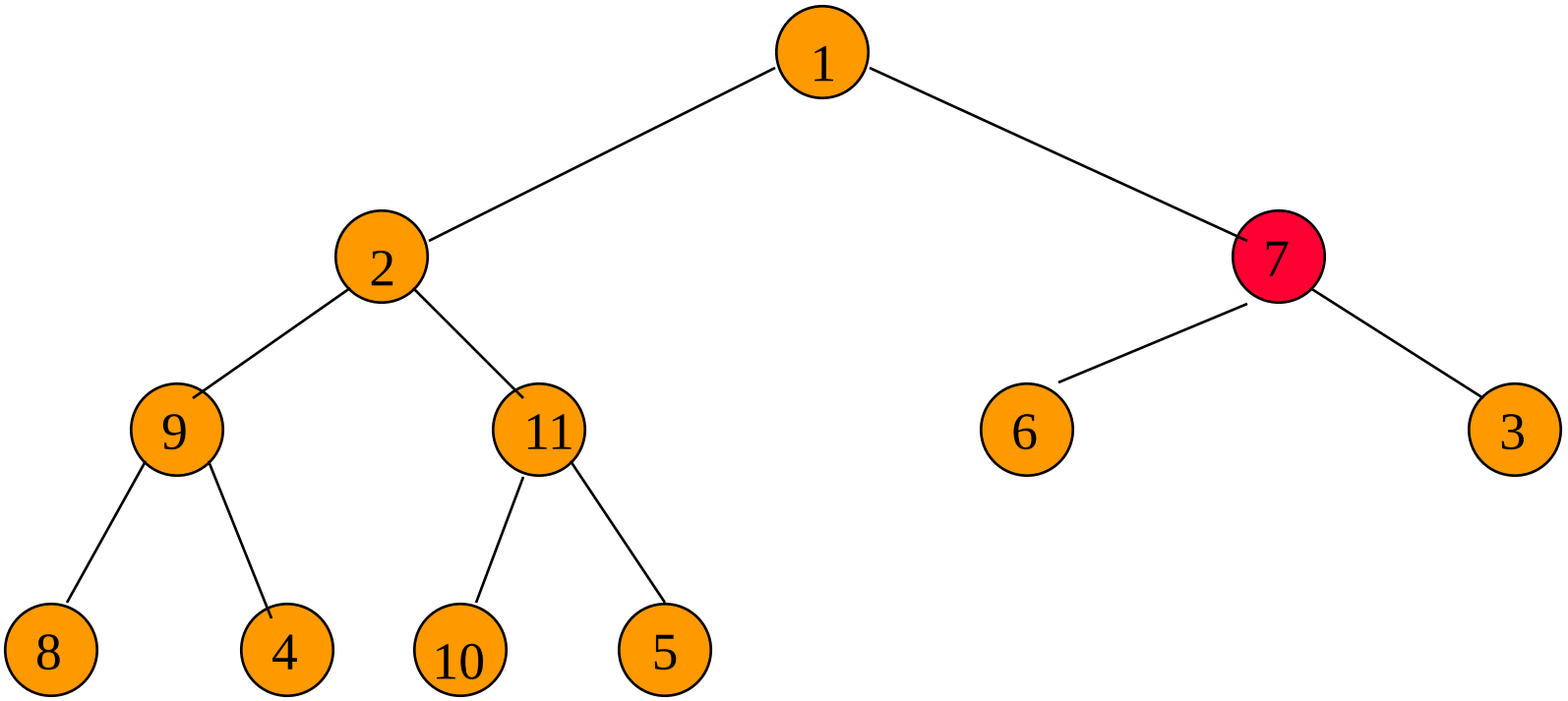
input array = [1, 2, 3, 9, 11, 6, 7, 8, 4, 10, 5]

Initializing A Max Heap



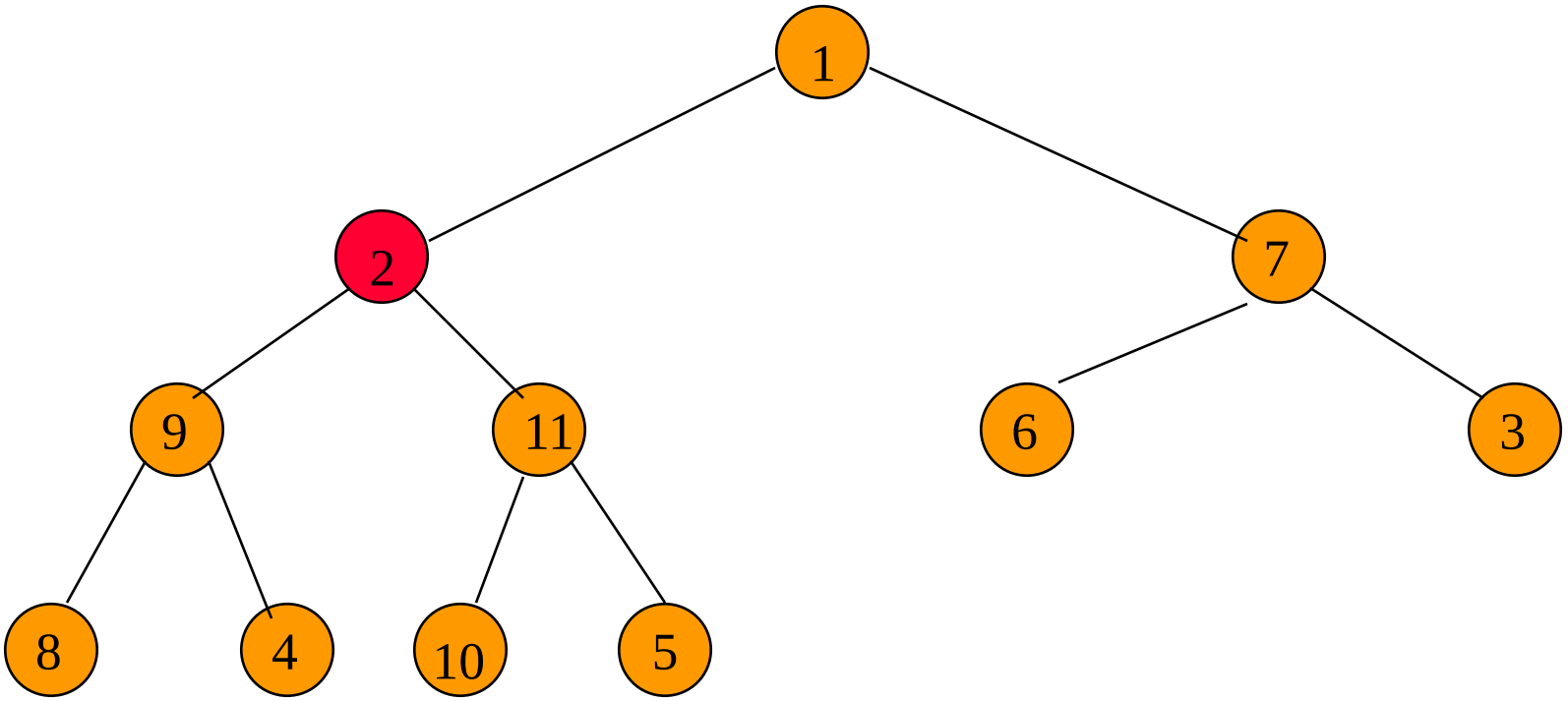
input array = [1, 2, 3, 9, 11, 6, 7, 8, 4, 10, 5]

Initializing A Max Heap



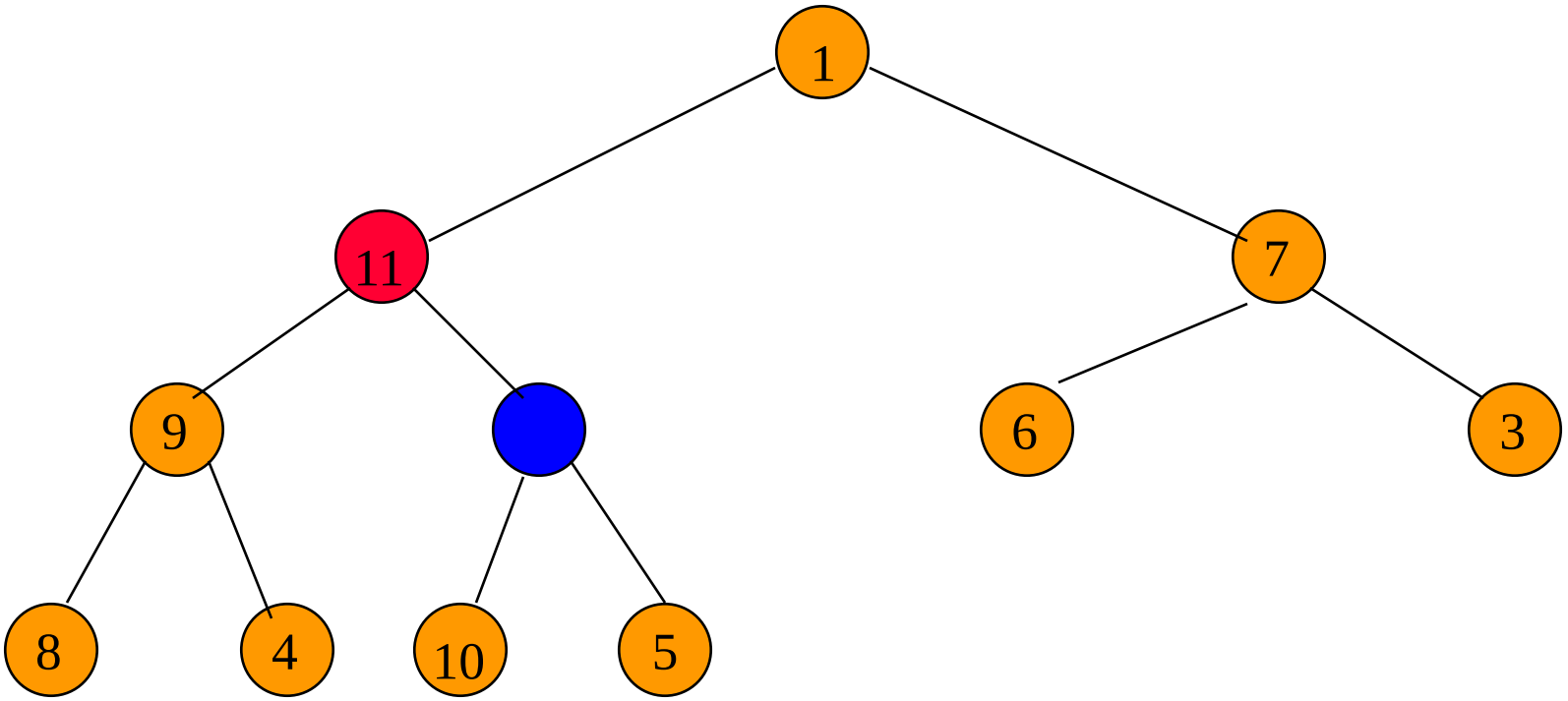
input array = [1, 2, 7, 9, 11, 6, 3, 8, 4, 10, 5]

Initializing A Max Heap



input array = [1, 2, 7, 9, 11, 6, 3, 8, 4, 10, 5]

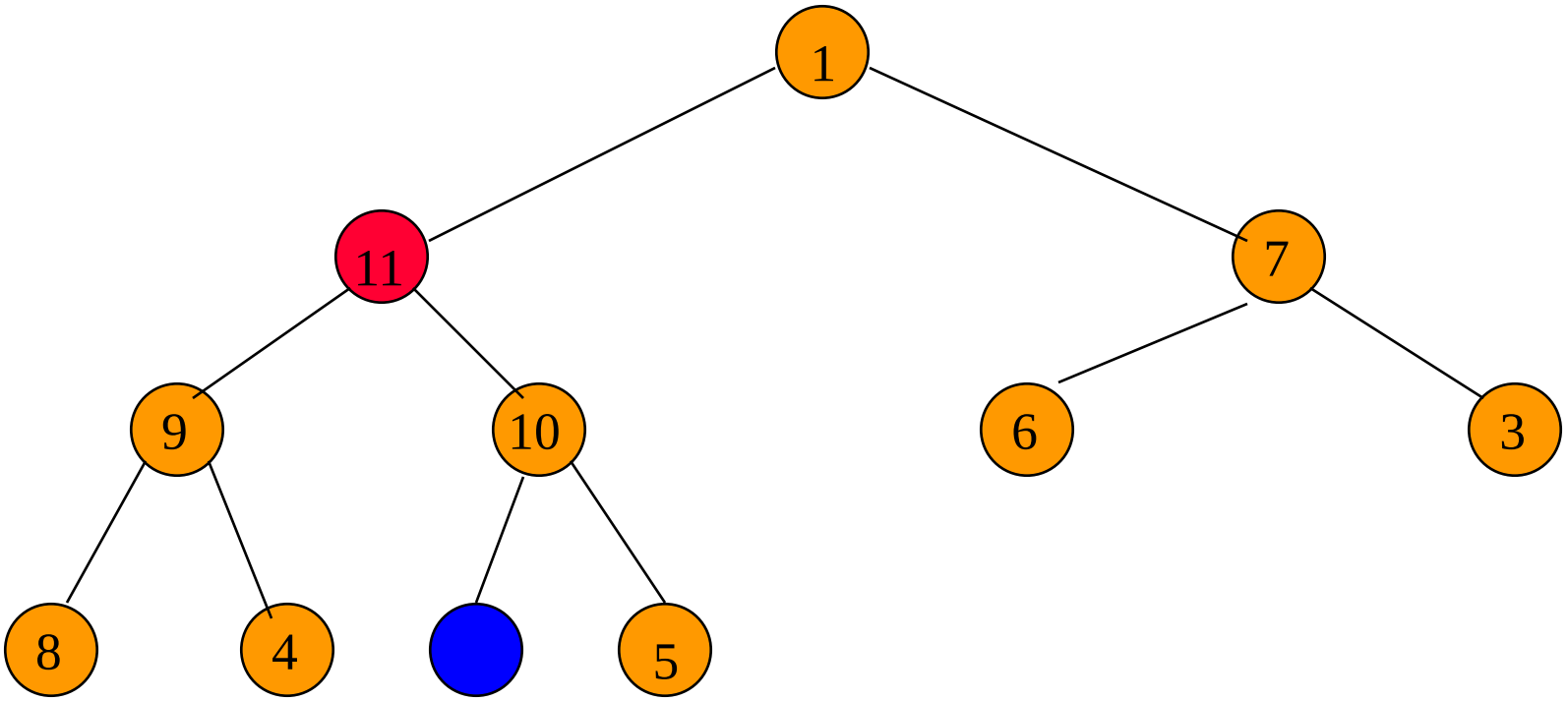
Initializing A Max Heap



input array = [1, 11, 7, 9, 2, 6, 3, 8, 4, 10, 5]

Find a home for 2.

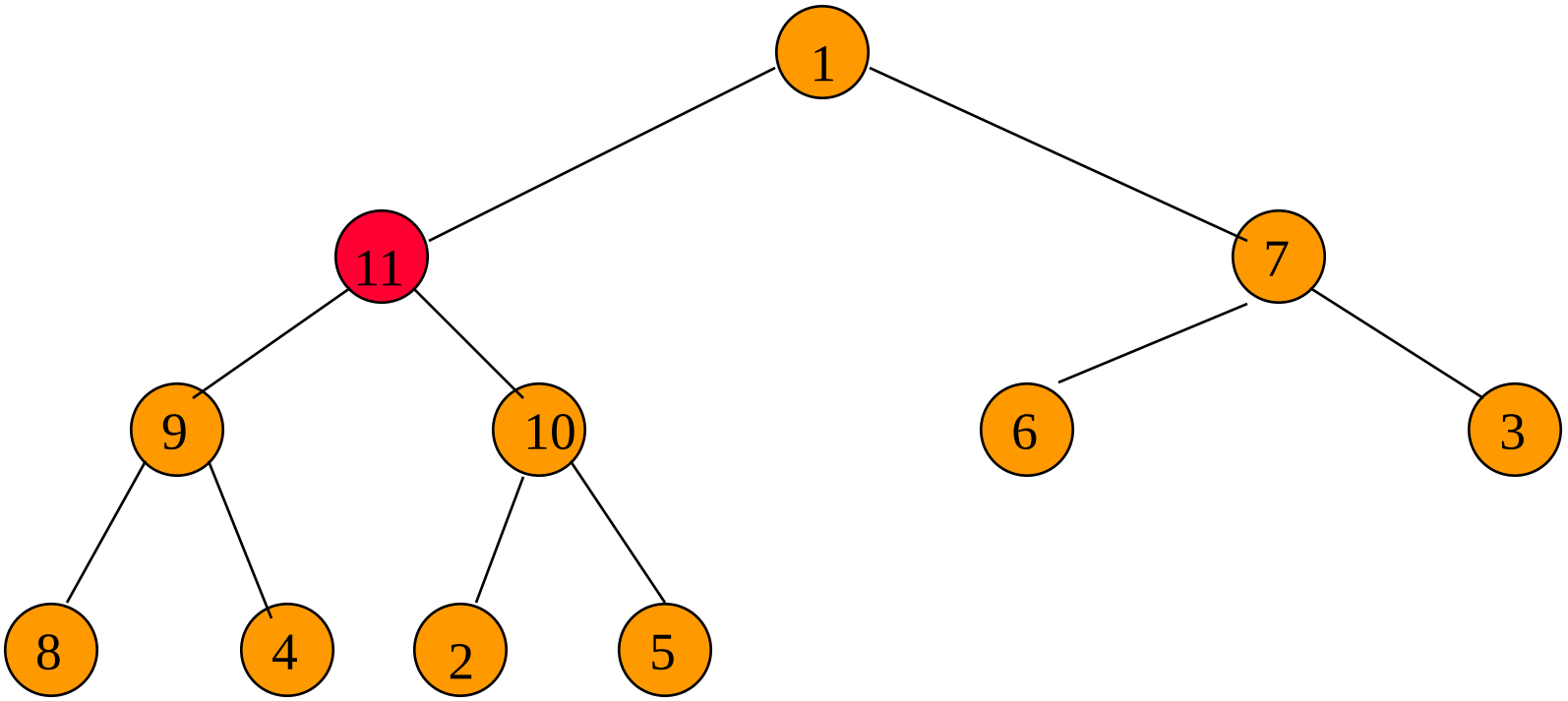
Initializing A Max Heap



input array = [1, 11, 7, 9, 10, 6, 3, 8, 4, 2, 5]

Find a home for 2.

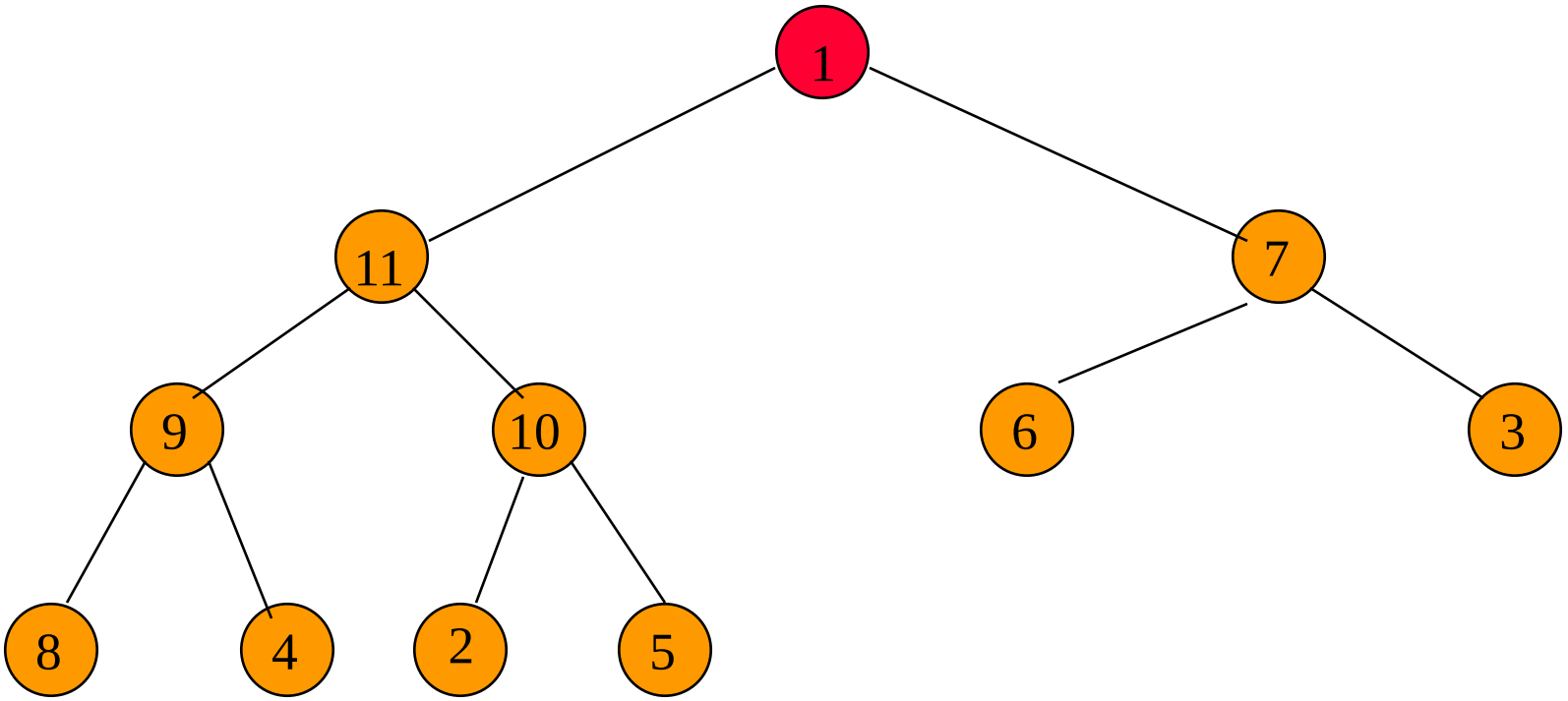
Initializing A Max Heap



input array = [1, 11, 7, 9, 10, 6, 3, 8, 4, 2, 5]

Done, move to next lower array position.

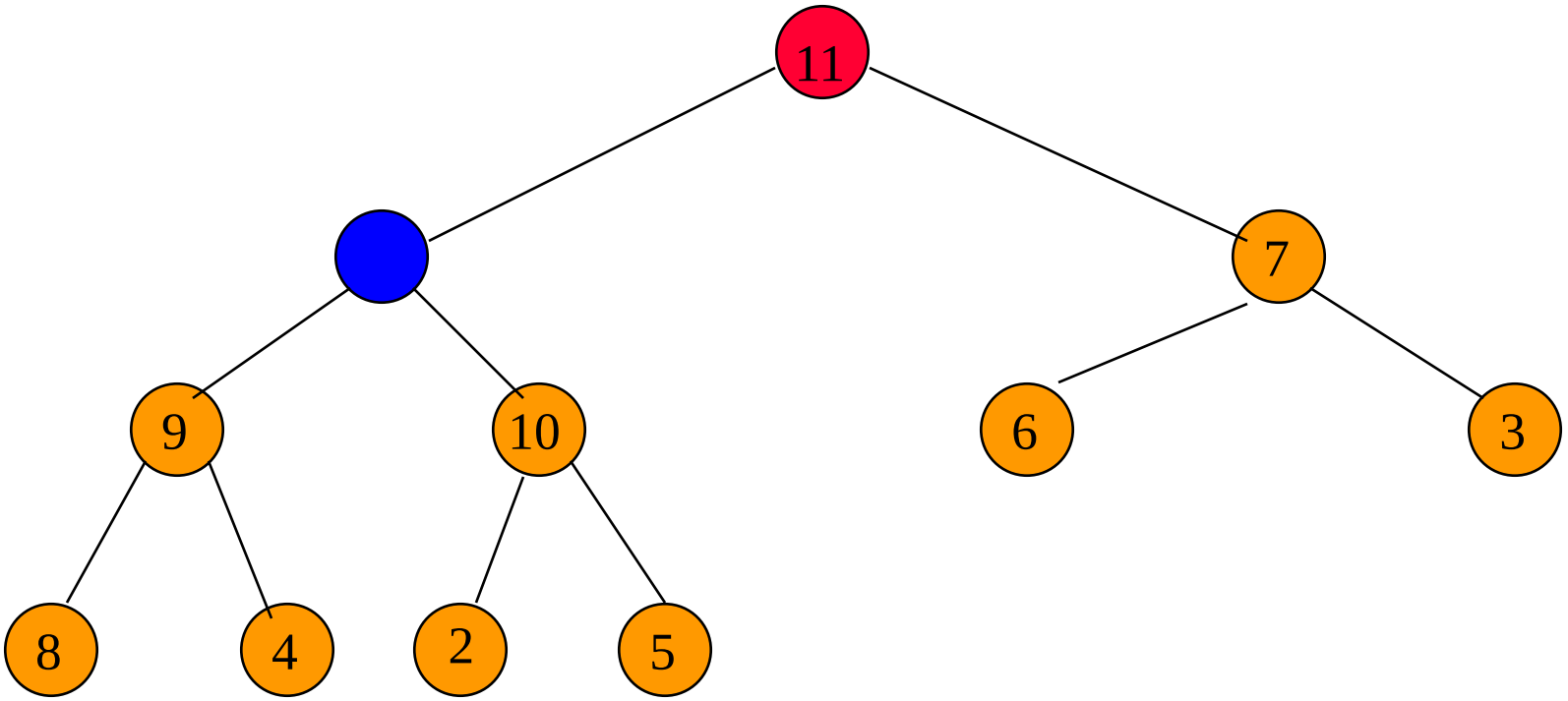
Initializing A Max Heap



input array = [1, 11, 7, 9, 10, 6, 3, 8, 4, 2, 5]

Find home for 1.

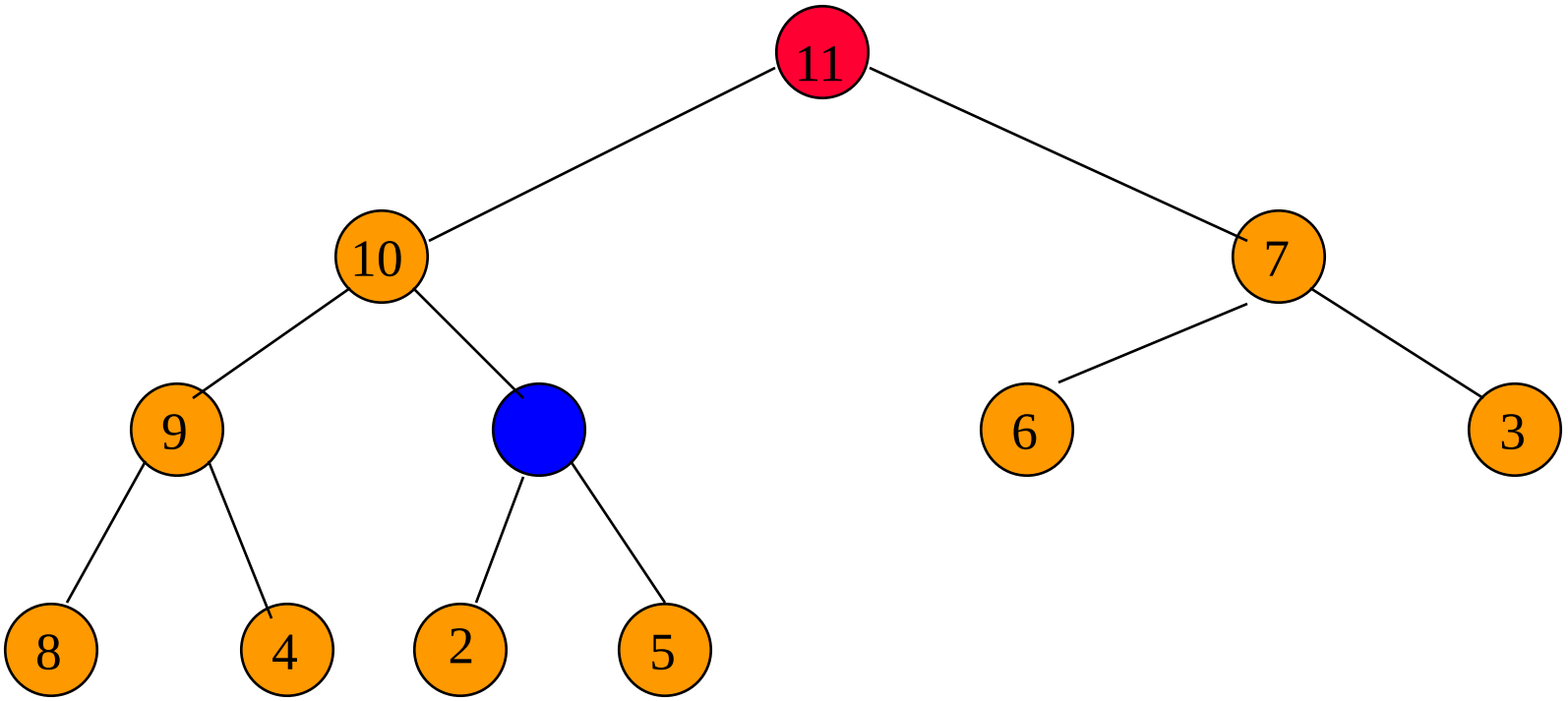
Initializing A Max Heap



input array = [11, 1, 7, 9, 10, 6, 3, 8, 4, 2, 5]

Find home for 1.

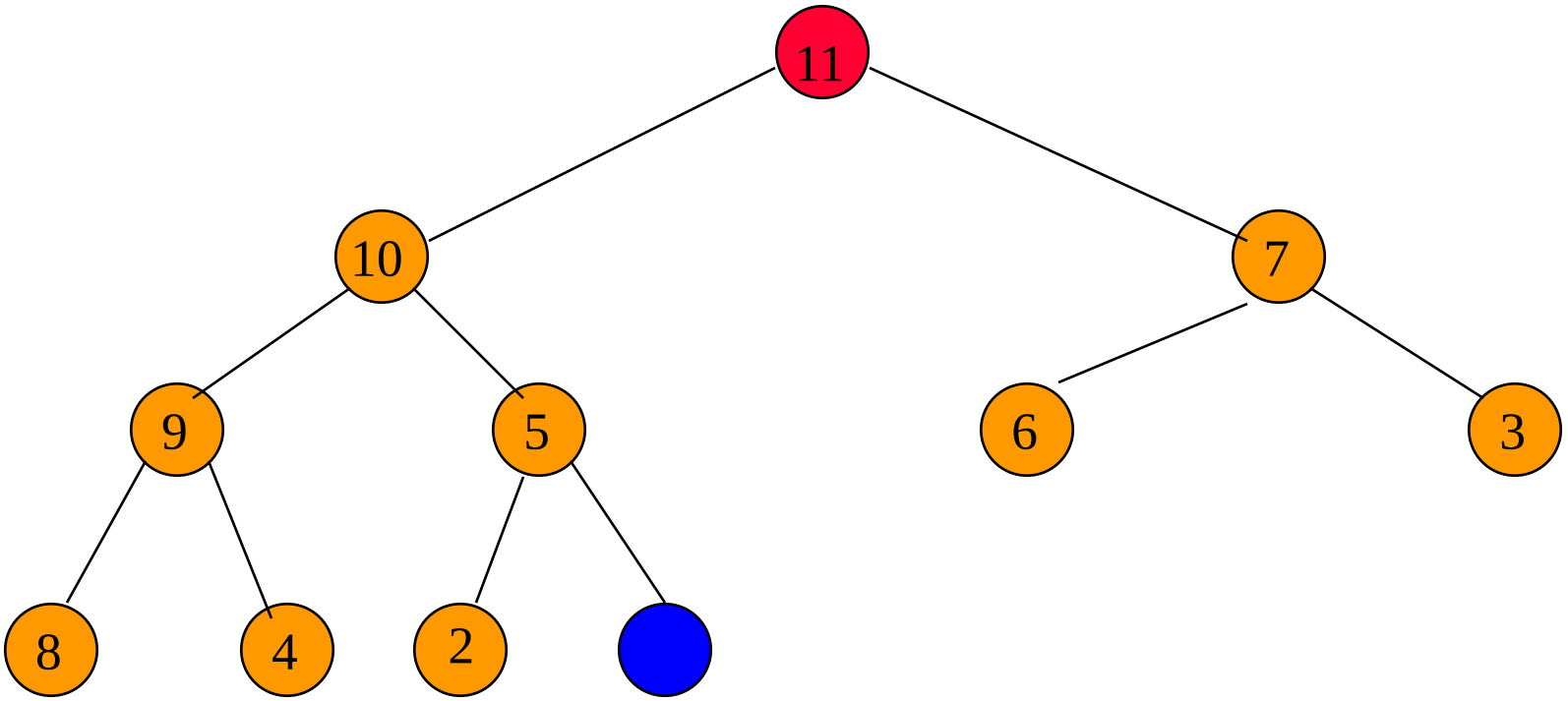
Initializing A Max Heap



input array = [11, 10, 7, 9, 1, 6, 3, 8, 4, 2, 5]

Find home for 1.

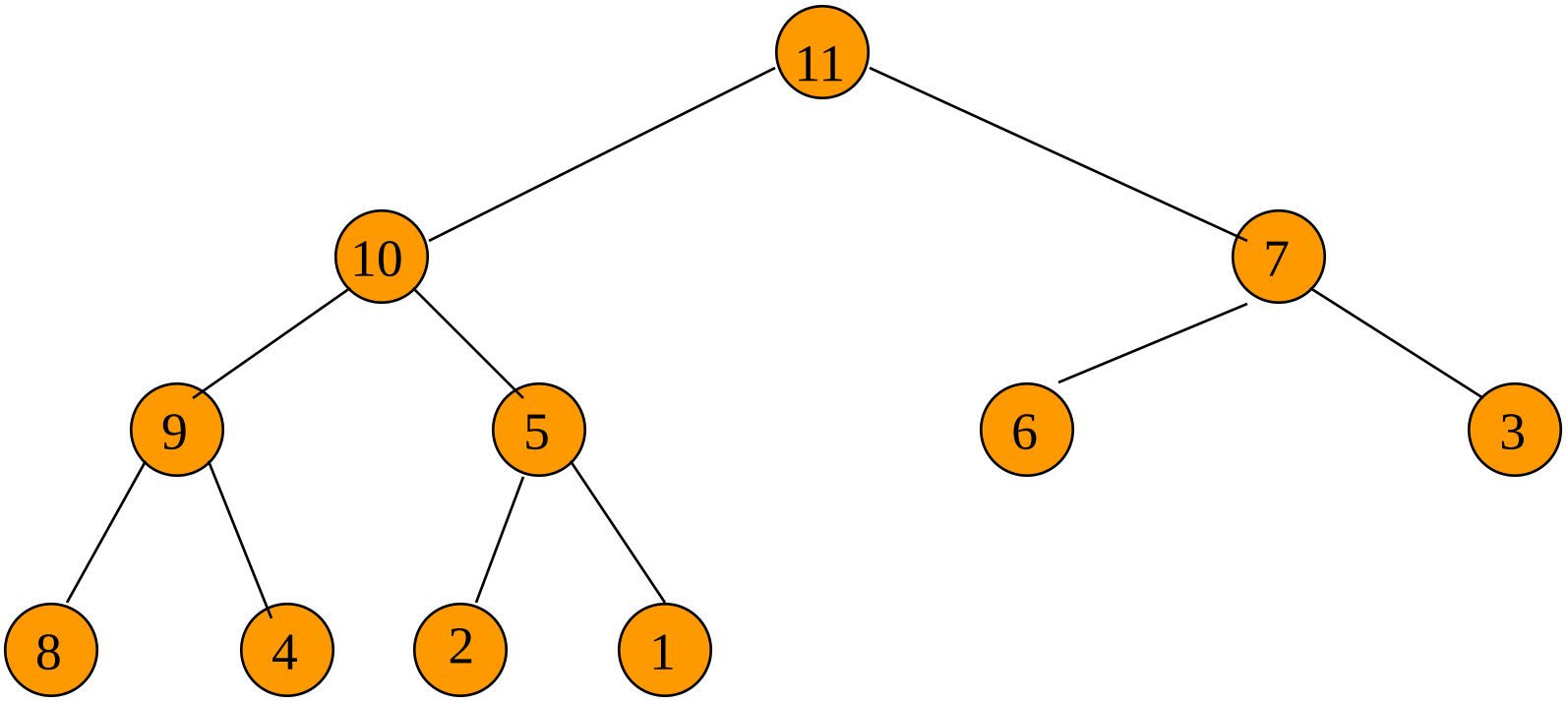
Initializing A Max Heap



input array = [11, 10, 7, 9, 5, 6, 3, 8, 4, 2, 1]

Find home for 1.

Initializing A Max Heap

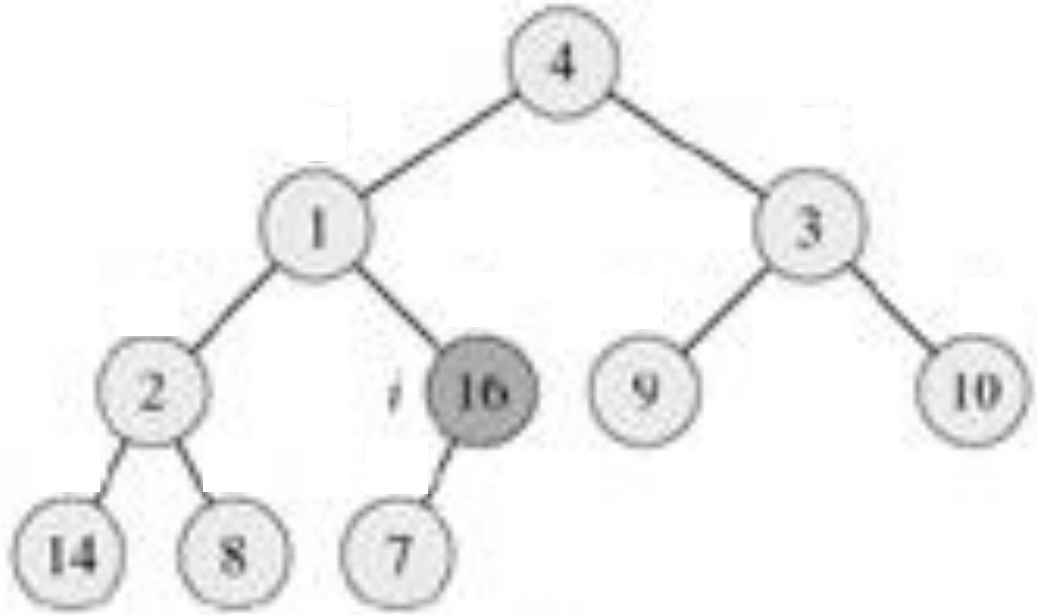


input array = [11, 10, 7, 9, 5, 6, 3, 8, 4, 2, 1]

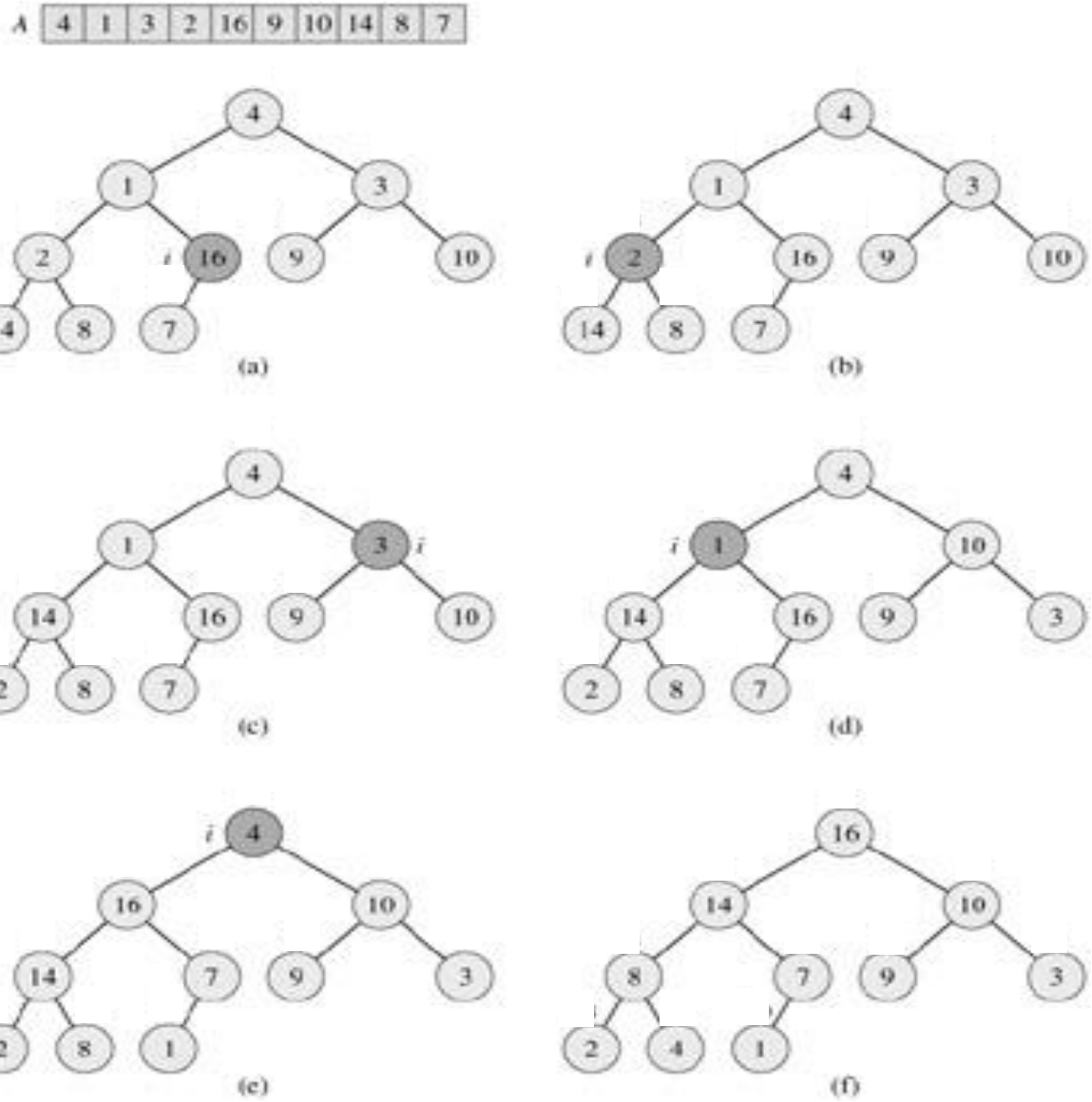
Done.

Initializing A Max Heap

A	4	1	3	2	16	9	10	14	8	7



Initializing A Max Heap



MaxHeap initialize

```
#initialize max heap to element array Heap
def initialize(self, heap, size):
  self.size = size
  for i in range((self.size-2)/2, 0):
    self.maxHeapify(i)
```

Heap Sort

Uses a max priority queue that is implemented as a heap.

http://www.youtube.com/watch?v=WYII2Oau_VY

Heap Sort

```
# sort the elements a[1 : a.length - 1] using the
heap sort method
def heapSort(a):
    # create a max heap of the elements
    h = MaxHeap(len(a))

    # extract one by one from the max heap
    for i in range(len(a) - 1, 0, -1):
        a[i] = h.removeMax(a, i+1)
```

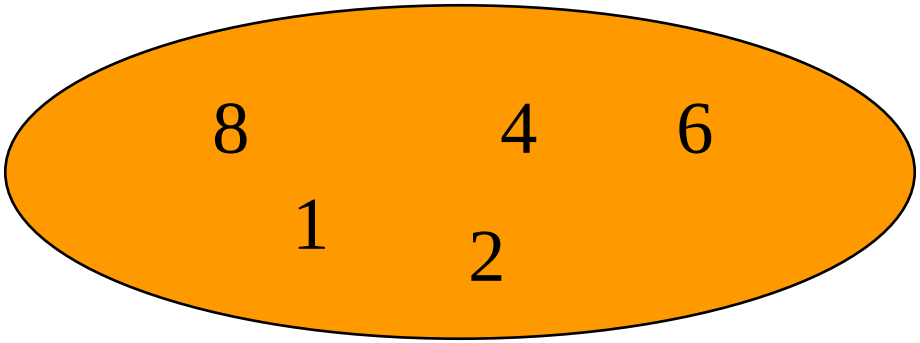
demo: HeapSort.py

Sorting Example

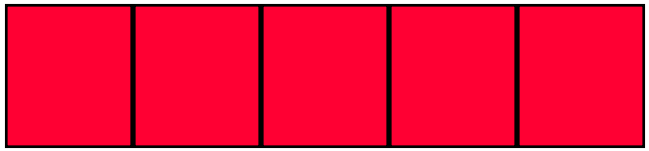
Sort five elements whose keys are 6, 8, 2, 4, 1 using a max priority queue.

- Put the five elements into a max priority queue.
- Do five remove max operations placing removed elements into the sorted array from right to left.

After Putting Into Max Priority Queue

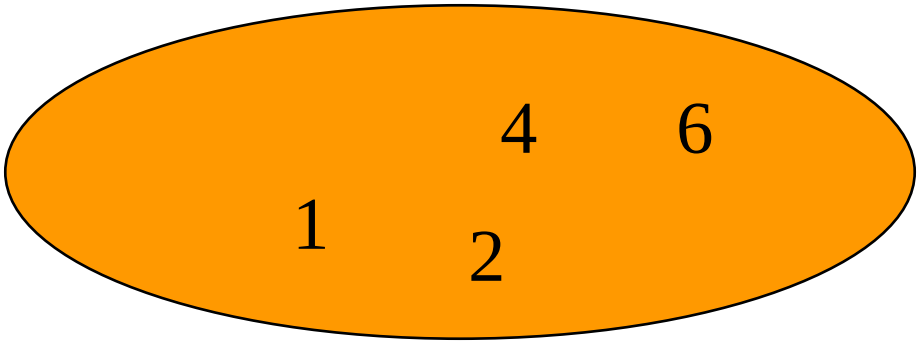


Max Priority Queue

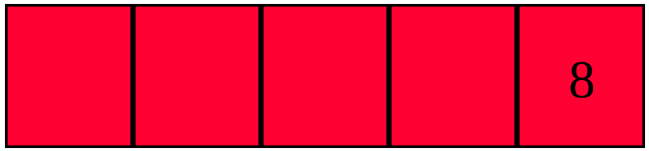


Sorted Array

After First Remove Max Operation

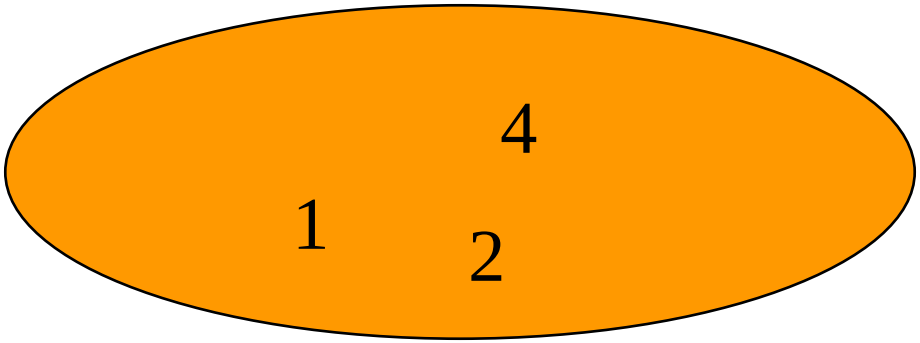


Max Priority Queue

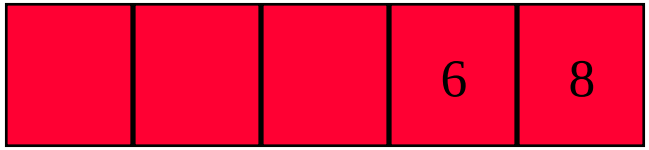


Sorted Array

After Second Remove Max Operation

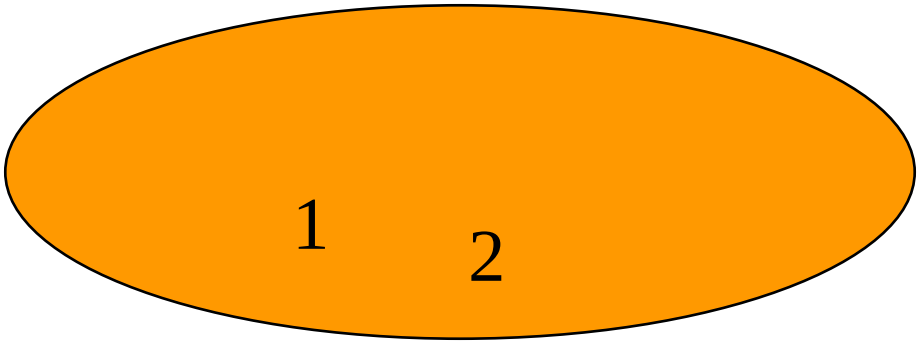


Max Priority Queue

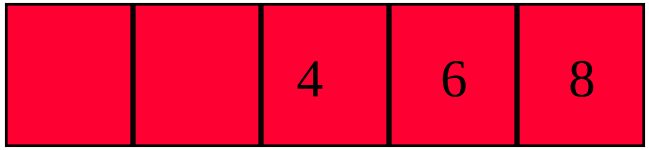


Sorted Array

After Third Remove Max Operation

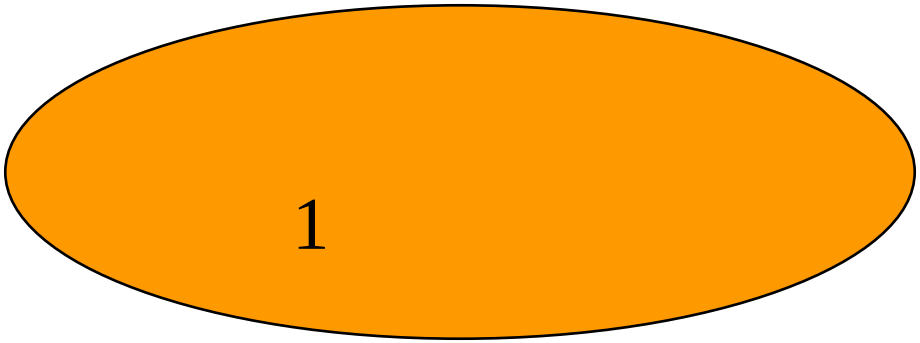


Max Priority Queue

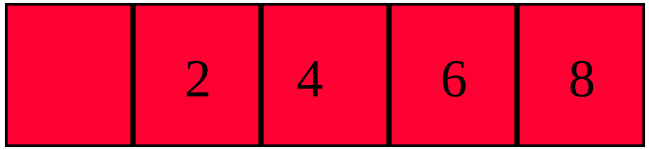


Sorted Array

After Fourth Remove Max Operation

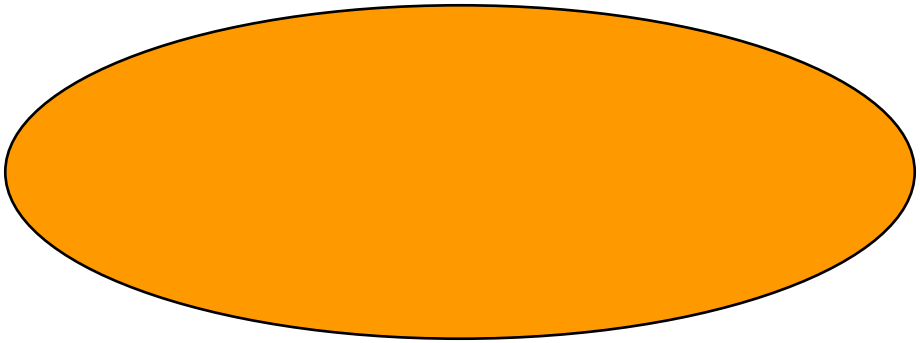


Max Priority Queue

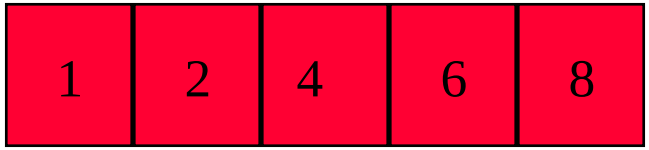


Sorted Array

After Fifth Remove Max Operation



Max Priority Queue



Sorted Array